TEKNISKA HÖGSKOLAN

Avdelningen för automations- och systemteknik

Johan Jansson

Integrering av informationssystem inom hälsovården – utveckling av en återanvändbar integreringsmodell

Diplomarbete inlämnat till granskning för diplomingenjörsexamen.

Karleby, 24.1.2005

Övervakare Professor Kari Koskinen
Datateknik inom automationen

Handledare FM Robert Åström
Oy Raisoft Ltd

TEKNISKA HÖGSKOLAN Avdelningen för automations- och systemteknik	SAMMANFATTNING AV DIPLOMARBETET

Författare Johan Jansson	Datum 24.1.2005
	Sidoantal 69 / 77

Arbetets titel Integrering av informationssystem inom hälsovården – utveckling av en återanvändbar integreringsmodell	
Professur Datateknik inom automationen	Kod AS-116
Övervakare Professor Kari Koskinen	
Handledare FM Robert Åström, Oy Raisoft Ltd	

Sammanfattning

Syftet med detta diplomarbete är att bygga upp en återanvändbar modell för integrering av hälsovårdssystem.

Först utförs en litteraturbaserad undersökning av systemintegrering för att få en överblick över hurudana tekniska lösningar som används inom systemintegrering, speciellt för integrering internt inom organisationer. Teknologier som används för integrering av hälsovårdsrelaterade informationssystem tas även upp.

Med denna undersökning som grund utvecklas sedan en återanvändbar modell för integrering av hälsovårdsrelaterade informationssystem. Målsättningen här är att ta fram en generell modell som kan tillämpas på flera olika typer av integreringslösningar. En liten försökstillämpning av denna modell implementeras för att utvärdera modellen. Försökstillämpningen verifierar att integreringsmodellen lämpar sig för återanvändning med olika integreringsteknologier.

Nyckelord
Applikationsintegrering, informationssystem inom hälsovården, återanvändbarhet

TEKNILLINEN KORKEAKOULU Automaatio- ja systeemitekniikan osasto	DIPLOMITYÖN TIIVISTELMÄ
Tekijä Johan Jansson	Päiväys 24.1.2005
	Sivumäärä 69 / 77
Työn nimi Terveydenhuollon informaatiojärjestelmien integraatio – uudelleenkäytettävän integraatiomallin kehitys	
Professuuri Automaation tietotekniikka	Koodi AS-116
Työn valvoja Professori Kari Koskinen	
Työn ohjaaja FM Robert Åström, Oy Raisoft Ltd	

Tiivistelmä

Tämän diplomityön tarkoituksena on kehittää uudelleenkäytettävää mallia terveydenhuollon tietojärjestelmien integraatiota varten.

Ensin tehdään kirjallisuustutkimus järjestelmäintegraatiosta. Tämän tutkimuksen tarkoituksena on saada yleiskuva siitä, millaisia teknisiä ratkaisuja on käytössä järjestelmäintegraatiossa, erityisesti organisaatioiden sisäisessä integraatiossa. Tutkimuksessa otetaan esiin myös ne teknologiat, jotka käytetään terveydenhuollon informaatiojärjestelmien integraatiossa.

Tämän tutkimuksen perusteella kehitetään sitten uudelleenkäytettävää mallia terveydenhuoltoon liittyvien informaatiojärjestelmien integraatioon. Tarkoitus on kehittää yleiskäyttöinen malli, jota voidaan soveltaa moneen eri tyyppiseen integraatioratkaisuun. Tästä mallista implementoidaan pieni testisovellus mallin arvioimista varten. Testisovellus vahvistaa mallin soveltuvuutta uudelleenkäyttöön eri integraatioteknologioiden kanssa.

Avainsanat
Sovellusintegraatio, terveydenhuollon informaatiojärjestelmät, uudelleenkäytettävyys

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Automation and Systems Technology	ABSTRACT OF THE MASTER'S THESIS
Author Johan Jansson	Date 24.1.2005
	Number of pages 69 / 77
Subject Integration of Information Systems in Healthcare – Development of a Reusable Integration Model	
Professorship Information Technology in Automation	Code AS-116
Supervisor Professor Kari Koskinen	
Instructor M.Sc. Robert Åström, Oy Raisoft Ltd	

Abstract

The purpose of this thesis is to develop a reusable model for integrating healthcare information systems.

First, a literature-based research is done to get an overview of the technical solutions used in application integration, especially in application integration within organizations. Technologies that are used for integrating healthcare-related information systems are also discussed here.

Based on this research, a reusable model for the integration of healthcare-related information systems is developed. The purpose is to create a general model that can be applied to different kinds of integration solutions. A small test application of this model is implemented to evaluate the model. The test application confirms that the model is reusable with different integration technologies.

Keywords
Application integration, information systems in healthcare, reusability

Förord

Diplomarbetet utfördes i samarbete med Oy Raisoft Ltd under år 2004 samt under början av år 2005. Jag skulle vilja rikta min tacksamhet till min professor Kari Koskinen och min handledare Robert Åström för den hjälp och det stöd de givit mig under arbetets gång. Jag vill också rikta ett tack till övrig personal på Raisoft för det stöd jag fått. Jag vill tacka mina föräldrar för den uppmuntran och det stöd de givit mig under studietiden. Mest av allt vill jag tacka min hustru Sanna för all den uppmuntran och hjälp hon givit mig under detta arbete.

Karleby, 24.1.2005

Johan Jansson

Innehållsförteckning

Förkortningar

API	Application Programming Interface
CCOW	Clinical Context Object Workgroup
CDA	Clinical Document Architecture
CIAS	Clinical Image Access Service
COAS	Clinical Observation Access Service
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
EAI	Enterprise Application Integration
EDI	Electronic Data Interchange
EDIFACT	EDI For Administration, Commerce, and Transport
HL7	Health Level Seven
IT	Informationsteknologi
LTC	Long-Term Care
MOM	Meddelandeorienterade mellanprogram
OMG	Object Management Group
PIDS	Person Identification Service
RAD	Resource Access Decision
RAI	Resident Assessment Instrument
RIM	Reference Information Model
RPC	Remote Procedure Call
TQS	Terminology Query Service
XML	Extensible Markup Language

1 Inledning

1.1 Bakgrund

Syftet med informationssystem inom hälsovården är att förbättra vården och att höja kvaliteten på vården. Med hjälp av modern informationsteknologi kan man effektivera behandlingen av vårdrelaterad information och på så sätt spara både tid och pengar.

Det finns dock inget informationssystem som täcker alla behov inom hälsovården. Ett typiskt scenario för användningen av informationssystem på en vårdanstalt är att man har ett så kallat patientadministrationssystem och ett flertal separata specialsystem. Patientadministrationssystemet upprätthåller patientregistret och de viktigaste patientuppgifterna. Specialsystemen används för mera specialiserat bruk, t.ex. laboratorie- och röntgensystem. Patientadministrationssystemet kan även ha inbyggda funktioner som fyller samma funktion som separata specialsystem. [Tervo-Pellikka 1996]

Emedan dessa olika system förbättrar kvaliteten på vården så kan det faktum att flera separata system används minska effektiviteten på vården. Vårdpersonalen är tvungen att mata in delvis samma patientuppgifter flera gånger i olika system. När personalen söker fram information om en viss patient kan man vara tvungen att öppna flera system och söka fram patientuppgifterna i alla dessa system innan man får fram all den information man behöver om patienten. Detta leder till att det går åt tid att göra samma sak i olika system. Samtidigt finns risken att informationen matas in fel i något system och att information om en patient i ett av systemen kan vara felaktig och även vara i strid med motsvarande information i ett annat system. [Tervo-Pellikka 1996]

I detta sammanhang spelar integrering en viktig roll för att effektivera användningen av informationssystem inom hälsovården. Med hjälp av integrering kan man till en viss del undvika situationer där vårdpersonalen är tvungen att upprepa samma arbete flera gånger, genom automatiskt utbyte av information mellan de olika informationssystemen.

Detta diplomarbete har utförts i samarbete med Oy Raisoft Ltd, ett företag som specialiserat sig på utveckling av programvara för RAI-bedömningssystemet inom åldringsvården. RAI (Resident Assessment Instrument) är ett internationellt använt system genom vilket man kan bedöma och uppfölja vården. Systemet är utvecklat för vårdplanering, kvalitetsuppföljning och kostnadsuppföljning. I Finland ansvarar Stakes (forsknings- och utvecklingscentralen för social- och hälsovården) samt Chydenius-Institutet för RAI-systemet, medan Raisoft utvecklar programvara för detta system. Raisoft grundades år 2000 och sysselsätter ett tiotal personer, företaget är verksamt i Karleby. Till kundkretsen hör såväl finländska som utländska hälsovårdsorganisationer. Företaget har utvecklat ett flertal programvaruprodukter inom RAI och av dessa kommer RAIsoft-LTC (Long-Term Care), en programvara för RAI-bedömningar inom långtidsvården, att användas som bas för detta diplomarbete. [RAI 2005]

1.2 Målsättning

Målsättningen med detta diplomarbete kan delas upp i två delar. Den första delen av målsättningen är att utföra en kartläggning av de integreringsteknologier som går att tillämpa för informationssystem inom hälsovården. Syftet med denna kartläggning är att få en överblick av läget inom systemintegrering, dels inom integrering av informationssystem överlag och dels inom integrering av hälsovårdssystem. Den andra delen av målsättningen är att utarbeta en modell för hur man kan utnyttja integreringsteknologier i RAIsoft-LTC programvaran. Denna modell kommer inte att vara en färdig lösning som kan tas i bruk som sådan. Syftet är att utarbeta en generell grundmodell på vilken man kan bygga de lösningar som sedan kommer att används i praktiken. Det innebär att planeringen sker till stor del på systemnivå och inte på detaljnivå. Till sist kommer jag att med en försökstillämpning pröva hur den utarbetade modellen fungerar i praktiken.

Integrering av informationssystem kan ske i olika stor omfattning, det finns integrering organisationer emellan samt integrering internt inom organisationer. I detta diplomarbete begränsar jag mig i huvudsak till integrering internt inom organisationer eftersom det är denna typ av integrering som är aktuell för Raisofts del.

1.3 Metodik

Detta diplomarbete kan ses som två delar, uppdelat enligt de två delmålsättningarna som ställts för detta arbete. Dvs. en kartläggning av integreringsteknologier samt planering av en modell för integrering av RAIsoft-LTC. För kartläggningen kommer jag att använda mig främst av litterära källor samt elektroniska publikationer. Jag kommer att börja med att kartlägga integrering av informationssystem inom organisationer. Efter det kommer jag att gå vidare till de standarder som används inom integrering av informationssystem för hälsovården.

Den andra delen, dvs. planering av en integreringsmodell för RAIsoft-LTC, kommer jag att utföra för Raisoft. Jag kommer att börja med att utarbeta en kravspecifikation och på basen av den planera själva modellen för hur RAIsoft-LTC kan integreras med andra informationssystem inom hälsovården. Till sist kommer jag att testa modellen med en försökstillämpning.

1.4 Uppläggning

I kapitel 2 tar jag upp integrering av informationssystem. Integrering inom organisationer behandlas med integrering av företagsprogram (EAI, eng. Enterprise Application Integration) som utgångspunkt. Bakgrund, integreringsnivåer, fördelar och problemområden samt teknologier behandlas. Syftet med detta kapitel är att ge en introduktion till integrering, med betoning på integrering inom organisationer.

I det tredje kapitlet tar jag upp standarder och gränssnitt för integrering. Både allmänt använda integreringsstandarder samt integreringsstandarder för hälsovårdssystem behandlas. Syftet för detta kapitel är att kartlägga de standarder för integrering som används inom hälsovården samt att visa vilken betydelse dessa standarder har för att möjliggöra integrering.

Kravspecifikationen för integreringsmodellen för RAIsoft-LTC kommer att behandlas i kapitel 4. I detta kapitel ställer jag upp en kravspecifikation för integrering av RAIsoft-LTC med övriga hälsovårdsapplikationer. Kapitlet ger även en kort överblick över RAIsoft-LTC, dess funktion samt hur programvarans användning kan förbättras genom integrering.

I det femte kapitlet utarbetar jag en modell för integrering av RAIsoft-LTC. Denna modell planeras alltså på basen av kravspecifikationen i kapitel 4. Syftet med den specifikation som ges i detta kapitel är inte att ge en komplett teknisk lösning som är färdig att tas i bruk, utan en specifikation på vilken man kan bygga upp och återanvända integreringslösningar.

I kapitel 6 redovisar jag för en försökstillämpning som implementerar modellen som beskrivits i kapitel 5. Här testas modellen för att se hur den fungerar i praktiken. Med hjälp av denna försökstillämpning utvärderas modellen.

Avslutningsvis kommer jag i kapitel 7 att dra att dra slutsatser, av integrering inom hälsovården överlag samt av den specifikation som utarbetats och testats i kapitlen 4 till 6. Här kommer även forskningsresultaten att lyftas fram samt förslag till vidareutveckling av det arbete som gjorts inom ramen för detta diplomarbete.

2 Applikationsintegrering på organisationsnivå – EAI

2.1 Inledning

Företag har använt sig av applikationer i årtionden. I dessa applikationer ingår bl.a. planerings-, produktions- och försäljningsapplikationer. Tidigare var dessa applikationer tekniskt sett isolerade från varandra. Avsikten med dem var endast att automatisera repetitivt arbete inom det område applikationen var avsedd för.

Så småningom insåg företagen vikten av applikationsintegrering. Att överföra data mellan två olika system manuellt är mer tidskrävande och öppet för uppkomst av fel än om data överförs automatiskt. Det fanns också ett behov av att i nyare informationssystem använda funktionaliteter i de system man redan hade i bruk. Dessa ovannämnda behov kunde man bemöta genom integrering av företagsprogram (EAI, eng. Enterprise Application Integration).

EAI är ett begrepp som är relaterat med mera än enbart teknologi. EAI är en process som engagerar både IT-personal och organisationens ledning. Ur teknisk synvinkel är EAI ett verktyg som möjliggör utbyte av data och funktionalitet applikationer emellan. Ur ledningens synvinkel är EAI ett verktyg som möjliggör automatisering av informationsflödet i organisationens affärsprocesser. EAI höjer effektiviteten inom organisationen genom att minska på de fel och brister som uppstår i informationsflödet samt genom att snabba upp informationsflödet i organisationens olika affärsprocesser.

Inom hälsovården finns det också många olika applikationer i bruk. I större hälsovårdsorganisationer använder man sig bland annat av planerings-, administrations-, patientadministrations- och laboratorieapplikationer [Tervo-Pellikka 1996]. Detta innebär att det inom denna bransch finns motsvarande behov för applikationsintegrering som inom företagsvärlden. Fastän EAI har utvecklats inom företagsvärlden så finns här principer som i det stora hela går att tillämpa även inom hälsovården. Den gemensamma nämnaren för applikationsintegrering inom företagsvärlden och inom hälsovården är att det är fråga om organisationer som

använder en mängd olika applikationer för sin verksamhet och har ett behov att integrera dessa applikationer med varandra.

2.2 Vad är EAI?

2.2.1 Allmänt om EAI

EAI är den process som innebär att man inom en organisation kopplar ihop olika applikationer och informationssystem. EAI är inte endast en teknisk lösning för att förflytta data från ett system till ett annat. EAI inbegriper en process för att hämta data från databaser och applikationer och omforma dessa data till en form som det mottagande systemet förstår. I denna process kan det även vara nödvändigt att tolka data, utföra dirigering av data, kontrollera integritet av data och kontrollera att dataflödet stämmer överens med en given affärsprocess. [Reese 2002]

EAI kan indelas i flera integreringsnivåer och beroende på hurudana behov som finns för ett EAI-projekt kan man betona olika nivåer. Enligt Gormly (2001) så kan EAI indelas i följande integreringsnivåer [Gormly 2001]:

- Plattformsnivån
- Integreringsstandardnivån
- Datanivån
- Applikationsnivån
- Affärsprocessnivån

Dessa integreringsnivåer beskrivs närmare i de följande styckena.

2.2.2 Plattformsnivån

Informationssystemen inom en organisation består oftast av ett flertal olika teknologier. Det kan finnas äldre system baserade på gammal teknologi och gammal hårdvara samt nya applikationer baserade på modern teknologi. Systemen kan variera från företagets egna skräddarsydda system till inköpta programvarupaket. Detta innebär att organisationen kan ha flera olika plattformar i bruk – olika hårdvara, olika operativsystem och olika mjukvaruteknologier.

Integrering på plattformsnivån är egentligen en förutsättning för EAI. På denna nivå innebär integreringen att man gör det möjligt för de olika systemen att kommunicera

med varandra. I de fall där systemen är baserade på olika plattformar är ett alternativ att bygga om systemen så att alla system körs på samma plattform. Detta är dock vanligen ingen vettig lösning, eftersom det kan bli mycket kostsamt att bygga om systemen. Dessutom blir man tvungen att anpassa alla framtida system till den valda plattformen. Ett bättre alternativ är använda sig av teknologi som möjliggör kommunikation mellan systemen oberoende av underliggande plattform. Detta kan t.ex. innebära konfiguration av operativsystem och nätverk. På denna nivå spelar även datasäkerheten en stor roll liksom optimeringen av kommunikationsförbindelserna [Gormly 2001].

2.2.3 Integreringsstandardnivån

För att olika applikationer ska kunna integreras så måste dessa olika applikationer kunna kommunicera på ett sätt som alla parter kan förstå. Detta innebär att man måste välja eller fastställa ett standardformat för dataöverföringen. Det finns flera olika teknologier som kan användas för detta ändamål, t.ex. COM+/DCOM, CORBA, EDI, JavaRMI och XML. [Gormly 2001]

2.2.4 Datanivån

En viktig del av EAI är utbytet av data applikationer emellan. De olika applikationerna som är i bruk inom en organisation lagrar sina data internt på olika sätt. För att kunna överföra data mellan de olika applikationerna så måste EAI-lösningen ha information om var data är lagrat i de olika applikationerna. Det vanligaste sättet att lagra data i moderna system är att använda relationsdatabaser. Det är dock inte sagt att alla applikationer använder samma teknik för att lagra data, organisationer kan även ha system där data lagras i t.ex. objektorienterade, multidimensionella, hierarkiska eller filbaserade databaser. [Linthicum 2001]

När man överför data mellan applikationer så räcker det inte med att helt enkelt läsa data från databas A och flytta eller kopiera det till databas B. I olika applikationer lagras data i olika strukturer som skiljer sig från varandra. T.ex. de data som representerar en beställning i en applikation skiljer sig från det data som representerar samma beställning i en annan applikation. Detta innebär att man måste ha en metadatamodell som beskriver hur en viss typ av information representeras i de olika applikationerna. Med andra ord måste man förstå vad olika data representerar

och på vilket sätt data är strukturerade. Med hjälp av en dylik modell kan man transformera data från en representation till en annan, så att de olika applikationerna får data enligt sina egna datarepresentationer.

De flesta moderna applikationerna har skiljt åt databasen från själva applikationslogiken, men det finns även system där applikationslogiken och databasen är tätt sammankopplade. I ett dylikt system räcker det inte till att känna till databasen och metadatamodellen för att kunna integrera applikationen; man måste även känna till applikationslogiken för att kunna integrera applikationen med de övriga applikationerna i organisationen. [Linthicum 2001]

Man måste alltså specificera på vilket sätt data skall transformeras från formatet i en applikation till formatet i en annan applikation. Detta sker på basen av metadatamodellen och applikationslogiken i de olika applikationerna. Integriteten av data är också en viktig aspekt i detta sammanhang. När man överför data från en applikation till en annan så måste man försäkra sig om att dataintegriteten bibehålls i båda applikationerna.

2.2.5 Applikationsnivån

Ett EAI-system behöver oftast mera än bara integrering på datanivån. Integrering på applikationsnivån innebär att man förutom att överföra data mellan applikationer också kan utnyttja de olika funktionaliteter som applikationerna erbjuder. Genom applikationsgränssnitt kan man komma åt applikationens funktionaliteter och det data som dessa funktionaliteter kan erbjuda. [Linthicum 2001]

Applikationsgränssnitt används alltså för att åstadkomma integrering på applikationsnivån. Ett applikationsgränssnitt (API, eng. Application Programming Interface), består av ett antal procedurer som program utanför själva applikationen kan kalla på. Med andra ord är ett API en mekanism för att få applikationen att utföra någon viss funktion, eller för att begära information av applikationen, utan att för den skull behöva använda applikationens användargränssnitt. Vissa applikationer kan erbjuda ett mycket omfattande API medan andra applikationer kan erbjuda ett väldigt begränsat API, eller inget API alls. Ett API kan erbjuda applikationens funktionalitet i olika sorters tjänster: funktionalitetstjänster, datatjänster och objekt.

Funktionalitetstjänster erbjuder ett gränssnitt till delar av applikationslogiken. Om applikationen som erbjuder funktionalitetstjänster har en applikationslogik som är tätt sammankopplad med data och databasen så kan applikationen erbjuda funktionalitetstjänster för att lägga till eller uppdatera data. På det viset behöver inte övriga program ha information om på vilket sätt data är kopplat till applikationslogiken. Applikationen sköter internt om att data är konsistent med applikationslogiken. T.ex. kan applikation A erbjuda en funktionalitetstjänst för att tillägga en beställning. Applikation B, som lägger till en beställning till applikation A, behöver då bara veta beställningens nummer, artiklar, datum, etc. Applikation B behöver inte veta på vilket sätt applikation A lagrar beställningen i databasen och hur beställningen påverkar applikationslogiken i applikation A, detta sköter A själv om. [Linthicum 2001]

Datatjänster erbjuder ett direkt gränssnitt till applikationens databas. Denna funktionalitet påminner om integrering på datanivån. Det är dock möjligt att i denna tjänst bygga in begränsningar, som vanligtvis inte är möjliga vid integrering på datanivån. Dessa begränsningar kan vara t.ex. integritetskontroll av data eller att hindra att man kan skriva data, alltså att utomstående program endast kan läsa data. [Linthicum 2004]

Objekt erbjuder samma funktionalitet som funktionalitets- eller datatjänster. Skillnaden ligger i att medan funktionalitetstjänster erbjuder procedurer och datatjänster erbjuder data, så erbjuder objekt både data och procedurer för att hantera data. Fördelen med objekt är att det finns färdigt definierade standardgränssnitt för dessa, så som CORBA och Java. [Linthicum 2004]

2.2.6 Affärsprocessnivån

Verksamheten i en organisation kan beskrivas som affärsprocesser. En affärsprocess är en serie arbetsskeden som utförs för att åstadkomma ett slutligt mål. Denna serie av arbetsskeden är något som sker gång på gång inom organisationens verksamhet, så som att ta emot en beställning och behandla den. [Grudén 2003] I de tidigare beskrivna integreringsnivåerna, mer specifikt datanivån och applikationsnivån, är fokus på dataflödet applikationer emellan. Detta har varit den traditionella

utgångspunkten inom applikationsintegrering men den tar dock inte organisationens affärsprocesser i beaktande.

När man bildar en affärsprocessmodell av verksamheten i en organisation, får man indirekt även en modell över hur data flödar i de olika affärsprocesserna. I utförandet av en affärsprocess ingår det flera applikationer och därmed även överföring av data applikationerna emellan. Dataflödet behöver dock inte nödvändigtvis följa samma mönster som processflödet, detta är illustrerat i bild 1.

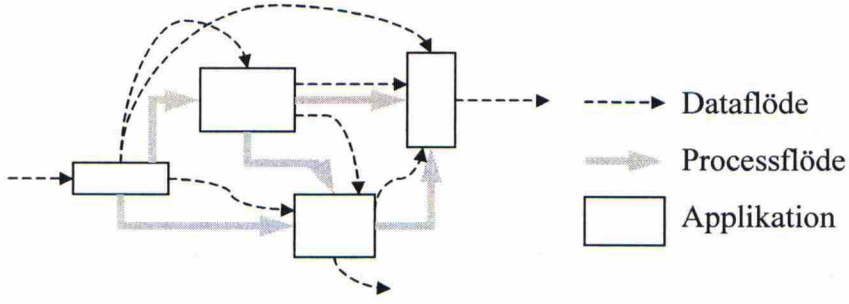

Bild 1 Dataflöde och processflöde applikationer emellan, inom en affärsprocess [Mann 1999].

När man utformar en modell av en affärsprocess skapar man bland annat regler för hur applikationer borde utbyta information. På affärsprocessnivån ligger integreringens fokus på att planera dessa regler på ett sätt som utnyttjar applikationerna så effektivt som möjligt. Detta innebär att affärsprocessintegrering borde vara verksamhetsorienterat till skillnad från de övriga integreringsnivåerna som är teknologiorienterade. [Linthicum 2001]

Integrering på affärsprocessnivån ger information om vilka applikationer som används i de olika skedena av processerna. Integreringen ger också information om hurudan information behövs, samt när den behövs, för de olika processerna. Man kan följa med i vilket skede varje process är och detta gör det möjligt att automatisera processer som berör flera olika applikationer. Detta innebär att organisationer kan förbättra sina processer och göra dem snabbare och mera pålitliga med hjälp av affärsprocessintegrering. [Grudén 2003]

2.3 Mellanprogram

2.3.1 Introduktion till mellanprogram

Mellanprogram (eng. middleware) är en teknologi som ofta används inom EAI. Den används för att koppla ihop olika applikationer med varandra, oftast genom att vidarebefordra data mellan applikationerna. Det finns många olika modeller som används för att vidarebefordra data mellan applikationer, nedan är en lista på de vanligaste modellerna enligt Linthicum (2004):

- Punkt-till-punkt

  Denna teknik innebär att man kopplar samman applikationer två och två. Denna koppling går inte att utöka till fler applikationer, utan när man lägger till en ny applikation i integreringen så måste man göra enskilda punkt-till-punkt förbindelser till var och en av applikationerna man vill ha förbindelse med. Denna teknik är mycket enkel och lämpar sig för små lösningar med få applikationer. Den lämpar sig inte för system där man har många applikationer i bruk, eftersom antalet punkt-till-punkt förbindelser växer mycket snabbt vilket leder till att kommunikationen applikationer emellan inte sker effektivt. [Linthicum 2004]

- Många-till-många

  Denna teknik innebär att man kan koppla ihop flera än två applikationer genom samma förbindelse. Dvs. man använder sig av ett förmedlarsystem mellan applikationerna. På så sätt räcker det att man gör endast en ny förbindelse när man lägger till en ny applikation. Denna modell lämpar sig bättre än punkt-till-punkt modellen till att använda i större system med många applikationer. Nackdelen är dock att systemet kan bli komplicerat med många olika förbindelser som skall administreras. [Linthicum 2004]

- Synkron/asynkron kommunikation

  Asynkron kommunikation innebär att när applikation A sänder ett meddelande till applikation B, så kan A fortsätta köra normalt utan att vara beroende av hur applikation B behandlar meddelandet. Applikation A kan dock vänta på svar från applikation B, men detta hindrar inte A från att utföra andra uppgifter. Synkron kommunikation i sin tur är blockerande. Detta innebär att när applikation A sänder ett meddelande till applikation B, så

väntar A på att B behandlar meddelandet och ger ett svar innan A kan fortsätta köra. Dvs. applikationerna är mer självständiga när man använder asynkron kommunikation medan synkron kommunikation medför att applikationerna blir mer beroende av varandra. [Linthicum 2004]

- Förbindelseorienterad och förbindelselös kommunikation

  Förbindelseorienterad kommunikation innebär att två applikationer upprättar en förbindelse, utbyter information och avslutar förbindelsen. Detta kan ske asynkront eller synkront. Förbindelselös kommunikation innebär att applikationerna inte upprättar en förbindelse, utan att en applikation skickar information till en annan applikation utan att behöva förvänta sig något svar. [Linthicum 2004]

- Direkt kommunikation/köad kommunikation

  Direkt kommunikation innebär det att information överförs direkt från avsändarapplikationen till mottagarapplikationen. I köad kommunikation skickar avsändarapplikationen informationen till en meddelandekö, därifrån meddelandena vidarebefordras till mottagarapplikationen av en köhanterare. Fördelen med köad kommunikation framför direkt kommunikation är att mottagarapplikationen inte behöver vara igång, eller att kommunikationsförbindelsen mellan avsändar- och mottagarapplikationerna inte behöver vara öppen i då informationen sänds, det räcker med att kommunikationsförbindelsen till köhanteraren är öppen. [Linthicum 2004]

- Utdelning/beställning

  Denna modell fungerar så att en applikation kan dela ut information utan att behöva veta något om mottagarapplikationerna. Informationen sänds till ett förmedlarsystem som i sin tur delar ut informationen åt de applikationer som beställt denna typ av information. [Linthicum 2004]

- Begäran/svar

  I denna modell kan en applikation skicka en begäran om viss information till en annan applikation och mottagarapplikationen skickar ett svar på denna begäran. [Linthicum 2004]

- Skicka-och-glöm

  Denna modell ger möjlighet åt en applikation att skicka iväg ett meddelande utan att behöva ta i beaktande vilka applikationer som mottar meddelandet eller svarar på meddelandet. Avsikten med denna modell är att ge möjlighet för applikationer att skicka ut meddelanden som berör många applikationer utan att behöva skicka ett skilt meddelande åt varje applikation. [Linthicum 2004]

Det finns även många olika typer av teknologier som man kan använda för mellanprogram. Enligt Linthicum (2004) är följande teknologier de vanligaste för mellanprogram:

- Fjärranrop av procedurer

  Fjärranrop av procedurer (RPC, eng. Remote Procedure Call) möjliggör en applikation att kalla på en procedur i en annan applikation på en annan dator. Denna procedur utförs synkront, vilket innebär att den kallande applikationen blockeras medan proceduren utförs i den andra applikationen. Fördelen med RPC är att det är en enkel teknik att använda. Nackdelen med RPC är att den kräver mycket resurser både i form av processorkapacitet och i form av nätverkskapacitet jämfört med ett lokalt utförande av en motsvarande procedur. Ett exempel på RPC är Open Software Foundations Distributed Computing Envinronment (DCE). [Linthicum 2004]
- Meddelandeorienterade mellanprogram (MOM)

  MOM kom till som ett svar på problemen i RPC. I MOM sker kommunikationen mellan applikationerna med hjälp av meddelanden, vanligtvis genom köad kommunikation. MOM är asynkront och blockerar således inte applikationerna medan kommunikationen är igång. MOM ger också en möjlighet att kommunicera applikationer emellan fastän applikationerna inte är igång samtidigt. Exempel på MOM är IBMs MQSeries och Microsofts MSMQ. [Linthicum 2004]
- Distribuerade objekt

  Med hjälp av distribuerade objekt kan man i en applikation köra ett objekt som använder metoder av ett likadant objekt som körs i en annan applikation. Detta sker så att man använder en objektstandard samt ett standardprotokoll

för kommunikationen objekten emellan. Fördelen med distribuerade objekt är att man kan skapa avancerade distribuerade system med hjälp av dem, nackdelen är att de kräver mycket resurser. De flesta distribuerade objektstandarder bygger på synkron kommunikation och är således blockerande. De två teknologier för distribuerade objekt som dominerar i dagens läge är Object Management Groups (OMG) Common Object Request Broker Architecture (CORBA) och Microsofts Component Object Model (COM). Man kunde även kalla Web Services för en typ av distribuerade objekt. [Linthicum 2004]

- Databasorienterade mellanprogram

  Databasorienterade mellanprogram fungerar som förmedlare mellan applikation och databas eller mellan två databaser. Det finns två typer av databasorienterade mellanprogram. Call-Level Interface (CLI) är ett generellt gränssnitt genom vilket man kan komma åt flera olika typer av databaser. Genom den andra typen, nativa gränssnitt, kan man å andra sidan enbart komma åt en viss sorts databas. Exempel på databasorienterade mellanprogram är Open Database Connectivity (ODBC) och Java Database Connectivity (JDBC). [Linthicum 2004]

- Transaktionsorienterade mellanprogram

  Transaktionsorienterade mellanprogram koordinerar informationsutbytet mellan applikationerna. Detta sker genom transaktioner, dvs. små arbetsenheter med en början och ett slut. Fördelen med dessa transaktioner är att man kan vara säker på att transaktionen endera slutförs eller så avbryts den och rullas tillbaka, dvs. transaktionen avslutas aldrig i ett halvfärdigt läge. Nackdelen med transaktionsorienterade mellanprogram är att informationsutbytet inte är lika effektivt som om det gjordes utan transaktionshantering. Användningen av transaktionsorienterade mellanprogram leder ofta till att man måste göra ändringar i de applikationer som integreras vilket i sin tur leder till att applikationerna blir tätt sammankopplade med varandra. Exempel på transaktionsorienterade mellanprogram är Microsoft Transaction Server (MTS), BEA Systems Tuxedo och IBMs Customer Information Control System (CICS). [Linthicum 2004]

2.3.2 Mellanprogram och EAI

Det enklaste sättet att utföra applikationsintegrering är att utföra punkt-till-punkt integrering. Detta innebär att applikationerna är direkt ihopkopplade. Ett dylikt system är lätt att implementera så länge antalet applikationer som integreras är lågt.

När man utför applikationsintegrering behöver man sätta upp regler för att kontrollera flödet av meddelandena mellan applikationerna, regler för vilka applikationer som behöver en viss information. Det behövs regler för dirigering, dvs. hur man ska vidarebefordra meddelandena så att rätt mottagare får meddelandet. Det behövs också regler för transformation av data mellan de dataformat som de olika applikationerna använder.

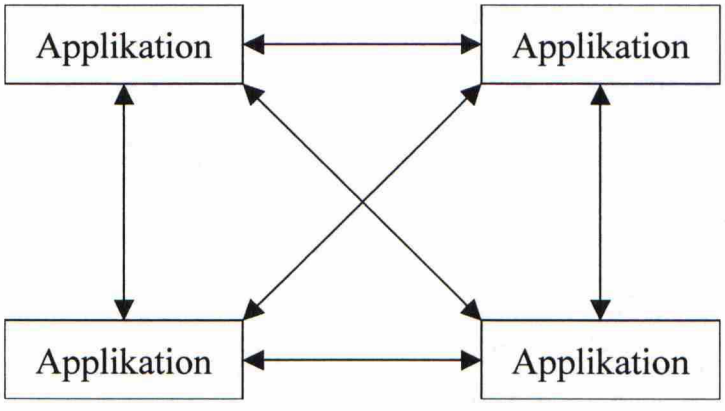

Bild 2 Punkt-till-punkt applikationsintegrering. Modifierad från [Mann 1999].

När det gäller punkt-till-punkt integrering så finns det inget separat förmedlarsystem mellan de olika applikationerna, detta är illustrerat i bild 2. Detta innebär att alla integreringsregler måste byggas in eller konfigureras för alla de olika applikationerna. När antalet applikationer ökar i den integrerade omgivningen, ökar komplexiteten i integreringsreglerna. Detta innebär att underhåll och administration av ett dylikt system blir en väldigt komplex uppgift ifall det finns många applikationer som integrerats. [Mann 1999]

För att lösa detta problem behöver man centralisera integreringsreglerna till ett ställe. Detta kan uppnås genom att använda ett förmedlarsystem och implementera integreringsreglerna i förmedlarsystemet. Förmedlarsystemet tar hand om

transformering av data och dirigering av meddelanden. Här kommer mellanprogram in i bilden, mellanprogram fungerar som förmedlarsystem för EAI. Med hjälp av mellanprogram är det enklare att administrera integreringsreglerna eftersom reglerna är implementerade endast på ett ställe i systemet. Med denna lösning behöver inte applikationerna kunna transformera data till alla de olika format som de övriga applikationer använder, det räcker att applikationerna vet hur data överförs till förmedlarsystemet. Detta är illustrerat i bild 3. [Mann 1999]

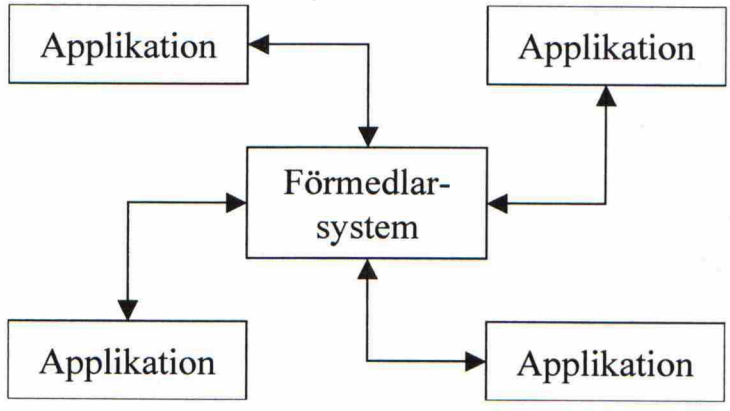

Bild 3 Förmedlad applikationsintegrering. Modifierad från [Mann 1999].

Mellanprogram gör det möjligt att dirigera information mellan applikationer enligt givna regler. Informationsflödet inom organisationer kan dock i vissa fall vara mycket komplex, så att enkla integreringsregler inte räcker till för att beskriva logiken i informationsflödet. Informationsflödet kan variera dynamiskt, t.ex. kan informationsflödet mellan en applikation som behandlar beställningar och övriga applikationer vara beroende av i vilket skede beställningen är.

I dylika situationer kan affärsprocessflödet vara till hjälp. Affärsprocessflödet utgör basen för informationsflödet och kan underlätta specificerandet av förmedlingssystemets integreringsregler. En lösning som följer av detta är att använda en processflödesapplikation för att specificera process- och informationsflödets regler och att låta mellanprogram utföra dirigering av data mellan applikationer i samarbete med processflödesapplikationen. Detta är illustrerat i bild 4. [Mann 1999]

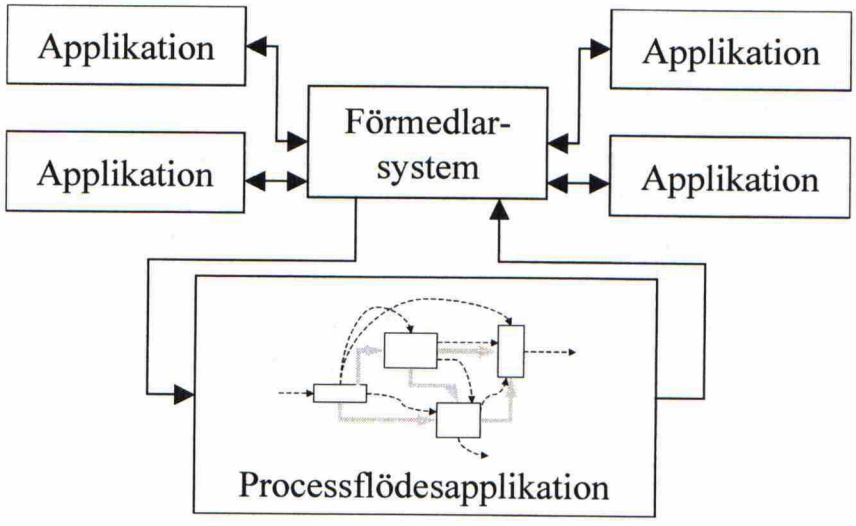

Bild 4 Processflödesorienterad integrering. Modifierad från [Mann 1999].

2.4 Adaptrar

En adapter inom applikationsintegrering är ett skikt mellan en applikation och ett mellanprogram, så som ett förmedlarsystem. Kommunikationen mellan applikationer och mellanprogram sker genom applikationens och mellanprogrammets applikationsgränssnitt. Dessa gränssnitt kan variera från väldokumenterade gränssnitt som är lätta att använda till komplexa gränssnitt som är svåra att använda. Med hjälp av adaptrar kan man gömma undan den komplexitet som finns i applikationsgränssnitten och transformera applikationens gränssnitt till mellanprogrammets gränssnitt. Utan adaptrar skulle man vara tvungen att skräddarsy kommunikationslänken mellan applikationen och mellanprogrammet för hand, adaptrar ger en möjlighet att återanvända mjukvaran som transformerar ett gränssnitt till ett annat. [Linthicum 2004]

Inom integrering kan adaptrar delas in i tunna och tjocka. En tunn adapter är egentligen bara en enkel programvara som översätter applikationens gränssnitt till mellanprogrammets gränssnitt. Fördelen med tunna adaptrar är att de är lätta att utveckla och lätta att ta i bruk. Nackdelen är att dessa adaptrar inte tillför någon mer funktionalitet, snarare syns effekten av dessa adaptrar i att de kräver mer resurser än t.ex. en skräddarsydd lösning. En tjock adapter i sin tur är en mer avancerad programvara som ger användaren möjlighet att koppla ihop olika applikationsgränssnitt genom ett grafiskt användargränssnitt, utan att behöva röra i

programkoden över huvudtaget. Dessa adaptrar kan ha inbyggd funktionalitet för kommunikation med olika typers mellanprogram, t.ex. distribuerade objekt eller meddelandeorienterade mellanprogram. Tjocka adaptrar är komplexa att utveckla och dyra i jämförelse med tunna adaptrar, men den funktionalitet som tjocka adaptrar medför ger dem en fördel över tunna adaptrar. [Linthicum 2004]

Man kan också särskilja statiska och dynamiska adaptrar. Statiska adaptrar måste konfigureras för hand, dessa adaptrar hämtar inte automatiskt in information om de omgivande systemen som kopplas samman av adaptern. Dynamiska adaptrar kan automatiskt lära sig om de system som adaptern kopplar samman. De kan även uppdatera den inlärda informationen när det sker en ändring i systemen som kopplas ihop av adaptern. En förutsättning för dynamiska adaptrar är att det finns till förfogande ett schema över hur data presenteras i de olika systemen som är kopplade till adaptern. [Linthicum 2004]

2.5 Varför använda sig av EAI?

I dagens läge använder organisationer sig i allmänhet av åtskilliga applikationer. Vissa applikationer kan vara färdiga produkter, köpta av återförsäljare, medan andra applikationer kan vara utvecklade eller skräddarsydda specifikt för organisationens egna behov. Somliga applikationer kan använda sig av ny teknologi medan det även finns äldre applikationer som baserar sig på gammal teknik.

De affärsprocesser som finns inom organisationen involverar ofta flera olika applikationer och detta leder till ett behov av att kunna överföra data mellan dessa olika applikationer. Traditionellt har denna överföring skett via människor, till exempel via telefon, fax eller e-post. Det faktum att människor fungerar som mellanhänder i överföringen av data medför vissa aspekter till dataöverföringen. Det uppkommer lätt ett fel i data som överförs, överföringen kan vara långsam emellanåt och det kan vara svårt att få en överblick över vilket tillstånd affärsprocesserna befinner sig i. [Reese 2002]

EAI erbjuder en lösning till detta problem. Med hjälp av EAI kan man automatisera det datautbyte applikationer emellan som följer vissa givna regler och på så vis kan fungera självständigt. Det är också möjligt att automatisera flödet i affärsprocesser.

På detta sätt kan organisationer med hjälp av EAI effektivera sina affärsprocesser, minska antalet fel i dataöverföringen och snabba upp informationsflödet. [Reese 2002] En följd av detta är att EAI ger mervärde åt applikationerna genom att integrera dem. Enligt Schmidt innebär detta att slutresultatet är större än summan av dess delar. [Schmidt 2002] Dvs. istället för att lösa problem på applikationsnivån kan man lösa problemen på processnivån. Detta tillvägagångssätt är viktigt eftersom dylika system tenderar att bli mer och mer komplicerade med tiden och om man fokuserar på applikationsnivån så blir det svårt att lösa problem som uppstår på processnivån.

I och med att EAI ökar på effektiviteten i affärsprocesserna inom organisationen kan EAI också ha en indirekt inverkan på organisationens kundrelationer. Organisationen kan med hjälp av EAI till exempel snabba upp eller höja kvaliteten på sina affärsprocesser och på så vis få mer nöjda kunder.

2.6 Problem med EAI

I det föregående stycket nämndes hur EAI kan lösa vissa problem relaterade med effektivitet inom organisationer. Trots det misslyckas cirka en tredjedel av alla EAI projekt [Farrell 2002]. Vad är då problemen med EAI? Varför leder inte EAI automatiskt till framgång?

Farrell (2002) har samlat de tio vanligaste orsakerna till att ett EAI-projekt misslyckas:

1. EAI-projektet är mer komplicerat och dyrt än man ursprungligen väntade sig.
2. De olika avdelningarna inom organisationen kommunicerar eller samarbetar inte med varandra. Traditionellt sett underhåller varje avdelning inom organisationen sina egna applikationer, medan EAI inbegriper applikationer från flera olika avdelningar inom organisationen.
3. Det finns inga standardmetoder för att bygga upp ett EAI-system, varje lösning är individuell.
4. Tillgång och användarrättigheter till information eller applikationer är inte klarlagda från början av projektet. EAI inbegriper många olika applikationer

och teknologier. Detta innebär att EAI-systemet måste ha tillgång till alla dessa olika system för att kunna integrera dem.

5. Ett EAI-projekt skiljer sig i stor grad från vanliga programvaruprojekt. Inom EAI måste man förstå de olika applikationerna för att kunna integrera dem. Det är också viktigt att förstå att EAI-projekt inte borde vara enbart teknologiorienterade. EAI inbegriper såväl en affärsverksamhetssynvinkel som en IT-synvinkel.
6. De olika applikationerna som ska integreras har olika datamodeller. Samma information representeras på olika sätt i olika system – detta gör kommunikationen mellan systemen komplicerad.
7. Man måste göra ett val mellan meddelandestorlek och återanvändbarhet. Om man strävar att återanvända data kommer det att ha en negativ inverkan på kommunikationerna i form av större meddelandestorlek.
8. Eftersom de flesta EAI-systemen bygger på asynkron kommunikation är det väldigt svårt att hantera transaktioner inom EAI-systemet. Detta innebär att dataintegriteten blir komplicerad att hantera.
9. Övervakning och administration av systemet i sin helhet krävs för att underhålla EAI-systemet. Detta är en krävande uppgift.
10. Det finns inga färdiga verktyg som är planerade för analysering av prestanda för ett EAI-system. [Farrell 2002]

2.7 Sammanfattning

EAI innebär en process som består av att automatisera informationsflödet mellan applikationer inom en organisation. EAI har alltså både teknologi och affärsverksamhet som utgångspunkt.

EAI kan hjälpa organisationen att förbättra sina affärsprocesser genom att optimera informationsflödet och på så sätt möjliggöra att processen utförs snabbare samt med färre fel. Med hjälp av EAI förbättras organisationens effektivitet och EAI kan på så sätt bidra till att organisationens kunder är mer nöjda med de tjänster eller produkter som organisationen producerar. Man kan dock inte anta att EAI är ett lätt sätt att förbättra organisationen. EAI är en komplicerad process som kräver mycket kunskap och arbete för att lyckas.

EAI består sällan av ett komplett paket som kan köpas av en återförsäljare. Det finns åtskilliga EAI-produkter på marknaden, dessa medför olika funktionaliteter som behövs inom EAI. För ett komplett EAI-system krävs det ofta flera olika EAI-produkter, till exempel ett förmedlarsystem samt en integreringsserver.

3 Integreringsstandarder

3.1 Introduktion till integreringsstandarder

Vid integrering av applikationer inom en organisation måste systemleverantörerna komma överens om hur information överförs från en applikation till en annan. I en liten omgivning med några få applikationer kan en lösning vara att skapa skräddarsydda kopplingar mellan applikationerna. Här kan man använda sig av olika tekniker så som att ha en överföringsfil som de olika applikationerna skriver till och läser ur med jämna mellanrum. En annan möjlighet är att applikationerna är i direkt kontakt med de övriga applikationernas databaser. Denna nivå av integrering kan jämföras med datanivån i kapitel 2.

Detta är dock en mycket kostsam verksamhet för systemleverantörerna, det tar mycket tid att skräddarsy en lösning för varje omgivning som produkten installeras i. Dessutom måste man utföra ändringar i alla de applikationer som är integrerade i omgivningen ifall det i ett senare skede kommer ändringar i lösningen. Ifall systemleverantörerna kunde återanvända en lösning, eller åtminstone en stor del av en lösning, så skulle man spara tid och resurser. Dessutom skulle administration och upprätthållning av de system som tagits i bruk underlättas ifall man återanvände lösningar, i och med att systemen är likadana, åtminstone till en del.

För att kunna återanvända en integreringslösning så bör man definiera ett gränssnitt genom vilket applikationen utbyter data med övriga applikationer. Denna nivå av integrering kan jämföras med applikationsnivån i kapitel 2.

En förutsättning för att man skall kunna återanvända en lösning för att koppla ihop applikationer är att även de övriga systemleverantörerna använder sig av en lösning som är kompatibel med den egna produkten. Det innebär att man måste använda sig av en standardiserad lösning ifall man inte vill skräddarsy en lösning för varje produkt som man integrerar. Med standardiserad lösning menas här såväl officiella standarder som inofficiella *de facto* standarder. När det gäller informationssystem för hälsovården kan man dela in integreringsstandarderna i två typer [Tervo-Pellikka 1996]:

- De standarder som är specifika för informationssystem inom hälsovården. Här avses standarder som är avsedda för att applikationer ska kunna utnyttja den information som överförs, som har med hälsovård att göra.
- De standarder som är allmänt i bruk för integrering av informationssystem. Här avses de generella standarder som används för att föra över data från ett system till ett annat. Dessa standarder används även inom andra områden förutom hälsovården.

3.2 Standarder specifika för hälsovården

I de nedanstående styckena tas upp några av de mest betydande standardiseringsorganisationerna och integreringsstandarderna som har direkt med hälsovård att göra.

3.2.1 Health Level 7

Health Level 7 (HL7) är en organisation som utvecklar standarder för integrering av informationssystem inom hälsovården. HL7 har specificerat ett antal meddelandeformat för att utbyta kliniskt och administrativt data mellan applikationer. Tanken bakom HL7 är att skapa ett gränssnitt för att utbyta information mellan applikationer för alla olika slags informationssystem inom hälsovården. Dvs. att man inte har begränsat sig till ett specialområde inom hälsovården, utan vill erbjuda en standard som kan användas inom alla hälsovårdens specialområden. Detta innebär att man borde kunna ta två eller flera olika hälsovårdsapplikationer som stöder HL7-gränssnittet och koppla ihop dessa utan att behöva göra några ändringar i applikationerna. [HL7 2004a]

HL7 grundades i USA år 1987 av en grupp sjukhus och deras systemleverantörer. HL7 är en icke-kommersiell organisation vars medlemmar består av intressegrupper inom hälsovården, så som organisationer som erbjuder hälsovård och tillverkare av informationssystem för hälsovård. [HL7 2004c] Dessa medlemmar står för organisationens verksamhet. Organisationen i USA står för utvecklingen av HL7-standarderna, denna organisation är en standardutvecklingsorganisation som är godkänd av American National Standards Institute (ANSI). Dessutom finns det nationella HL7-föreningar i övriga länder som är samarbetspartners med den amerikanska HL7-organisationen. HL7 Finland r.f. är en av dessa. HL7 har i dagens

läge blivit en av de största internationellt använda de facto standarderna för informationsutbyte för system inom hälsovården. I styckena nedan tas de viktigaste standarderna i HL7 version 3.0 upp. [HL7 2004a]

Referensmodell för information

Referensmodellen för information (RIM, eng. Reference Information Model) är en modell för den kliniska information som behandlas i de olika meddelanden och dokument som finns inom ramen för HL7. RIM innehåller en representation av klinisk information från alla olika områden inom hälsovården, på så sätt är RIM en grundmodell på vilken man baserar alla meddelanden och dokument för kliniskt data. RIM ger på så sätt en gemensam terminologi så att olika applikationer kan ha en gemensam definition på den kliniska information som behandlas i applikationerna. På så sätt kan man i en applikation ge klinisk information om en patient och när man via HL7-meddelanden överför denna kliniska information till en annan applikation, så kan den applikationen med hjälp av RIM veta vad denna kliniska information innebär. [HL7 2004a]

RIM är dokumenterad med hjälp av UML (Unified Modeling Language) och den innehåller sex stycken grundklasser, på vilken informationsmodellen byggs upp. Förutom klassdiagram så innehåller RIM även tillståndsdiagram, användningsfall, interaktionsdiagram, modeller för datatyper samt övriga modeller för att beskriva krav samt design av HL7 standarderna. På basen av denna grundmodell (RIM) bygger man sedan upp informationsmodeller för de olika specialområdena inom hälsovården. [HL7 2004d]

Arkitektur för kliniska dokument

Arkitekturen för kliniska dokument (CDA, eng. Clinical Document Architecture) är en standard för att utbyta kliniska dokument, så som olika rapporter angående patientens tillstånd [HL7 2004a]. Den version av CDA som beskrivs här är CDA R2 [HL7 2004b]. Ett CDA-dokument består av ett XML-dokument som innehåller klinisk information, dokumentet är sammansatt av både sådan information som kan visas direkt åt användaren och sådan information som är avsedd endast för applikationen. Ett CDA-dokument består av ett CDA-huvud samt en CDA-kropp. I huvudet finns en beskrivning av innehållet, dvs. information som identifierar

patienten, vårdperioden, vårdorganisationen, mm. Den kliniska informationen finns i kroppen av dokumentet, denna kan bestå av en binärfil eller alternativt en eller flera CDA-sektioner. I en CDA-sektion kan man dels lägga in information som är avsedd att visas åt användaren direkt (CDA Narrative Block), dels information som är avsedd åt den mottagande applikationen (CDA Entry). Informationen i CDA Entry följer RIM-modellen. [HL7 2004b]

Kontexthantering

Clinical Context Object Workgroup (CCOW) är en standard för kontexthantering hälsovårdsapplikationer emellan. CCOW utvecklades ursprungligen av en självständig organisation, men blev en del av HL7 standarderna år 1999. [HL7 2004a, Seliger 2001] Med hjälp av kontexthantering kan man synkronisera de olika applikationer som används av samma användare, så att alla applikationer automatiskt följer den kontext användaren arbetar med. På så sätt underlättas användarens arbete och all den information som finns att fås om patienten är lättillgänglig för användaren. [Seliger 2001]

Denna standard bygger på principen att man kan bestämma en kontext som är gemensam för de applikationer som användaren har i bruk. Denna kontext kan identifieras med hjälp av information t.ex. om användare och den patient vars uppgifter behandlas. [Seliger 2001]

Ur teknisk synvinkel består CCOW av tre komponenter [Seliger 2001]:

- De applikationer som användaren har i bruk.
- En kontexthanterare som synkroniserar kontexten mellan applikationerna. Applikationer kan både tilldela kontext till kontexthanteraren samt hämta kontextinformation från kontexthanteraren.
- Agenter[1] som kan konvertera kontextinformation från en applikation till ett format som en annan applikation kan förstå.

I praktiken fungerar kontexthanteringen på följande vis: Användaren har två eller flera applikationer igång som stöder CCOW. Användaren väljer kontext i någon av

[1] Med agent menas i detta sammanhang ett program som används av ett annat program.

applikationerna, t.ex. genom att ta fram en viss patients uppgifter på skärmen. Nu meddelar denna applikation åt kontexthanteraren om att användaren ändrat kontext och skickar kontextinformation, dvs. information som identifierar patienten, åt kontexthanteraren. Kontexthanteraren använder sig av en agent för att konvertera denna kontextinformation till sådan form som kontexthanteraren kan förstå, sedan uppdaterar kontexthanteraren sin egen information om kontexten samt meddelar åt de övriga applikationerna om att kontexten har ändrat. De övriga applikationerna hämtar kontextinformation från kontexthanteraren och visar automatiskt upp den valda patientens uppgifter i användargränssnittet. [Seliger 2001]

CCOW finns tillgängligt för flera olika teknologier. Version 1.0 av CCOW innehöll en specifikation för COM/ActiveX. Version 1.2 innehöll en specifikation för webbteknologier, där applikationerna kunde kommunicera genom meddelanden sänt över HTTP. I och med version 1.5 finns det även en specifikation för SOAP-protokollet. CCOW är alltså inte bundet till en viss teknologi, det går till och med att använda applikationer som stöder olika teknologier, så länge som kontexthanteraren stöder alla de teknologier som de olika applikationerna använder. [Seliger 2001]

3.2.2 OMGs arbetsgrupp för hälsovårdsrelaterade standarder

Object Management Group (OMG) är ett konsortium vars uppgift är att befrämja användandet av objektorienterad teknologi inom utvecklingen av distribuerade informationssystem. OMG grundades år 1989 och har över 800 medlemsföretag. Inom OMG verkar en arbetsgrupp för hälsovårdsrelaterade standarder, Healtcare Domain Task Force, även kallad CORBAmed. Denna arbetsgrupp har utvecklat ett antal standarder som tas upp kort i följande stycken. Dessa standarder definierar CORBA-gränssnitt för användning inom hälsovårdssystem. [OMG 2004, Culpepper 2000]

Tjänst för personidentifiering

Inom hälsovården lagras en patients uppgifter i flera olika informationssystem. Hälsovårdsorganisationer har oftast ett flertal informationssystem i bruk och dessutom kan patienten söka vård från flera olika hälsovårdsorganisationer under sin livstid. I vart och ett av dessa system associeras ett id med patientens uppgifter, detta

id identifierar patienten på ett unikt sätt inom den id-domän[2] som systemet använder sig av. De olika informationssystemen behöver dock inte använda sig av samma id-domän. Detta kan leda till problem när man vill integrera två system och överföra patientinformation från det ena systemet till det andra, eftersom patientens identifiering sker på olika sätt i de olika systemen. [OMG 1998b]

Tjänsten för personidentifiering (PIDS, eng. Person Identification Service) är en standard som möjliggör identifiering av patienter mellan olika informationssystem. Med PIDS kan man koppla samman flera id-domäner till en korrelerad id-domän, id:n kan korreleras mellan de olika domänen manuellt eller automatiskt. Med hjälp av PIDS kan man även söka patientuppgifter enligt ett givet id eller vice versa. PIDS stöder även sättandet av en policy för konfidentiella uppgifter, denna policy kan bero på användarens identitet, patientens identitet, id-domänen samt patientens uppgifter. [OMG 1998b]

Tjänst för terminologiförfrågningar

Informationssystem inom hälsovården används bland annat för att lagra klinisk information. För att kunna lagra denna information så behövs det ett lexikon, eller ett terminologisystem, med vars hjälp man kan entydigt beskriva denna kliniska information och lagra den i informationssystemet. När det gäller integrerade system så kan fallet vara att applikationer är tvungna att använda sig av två eller flera olika terminologisystem, vilket betyder att applikationerna måste vara anpassade till vart och ett av de gränssnitt som de olika terminologisystemen erbjuder. [OMG 1998a, Culpepper 2000]

Tjänsten för terminologiförfrågningar (TQS, eng. Terminology Query Service), är en standard som erbjuder ett gemensamt gränssnitt för förfrågningar till terminologisystem [OMG 1998a, Culpepper 2000, OMG 2004]. Det gränssnitt som TQS specificerar är alltså inte beroende av den underliggande implementationen av terminologisystemet. TQS bygger på element som är allmänt använda i terminologisystem och på basen av dessa element har man byggt upp ett gränssnitt

[2] Med en id-domän avses ett sätt att identifiera personer unikt, dvs. när man tilldelat ett id åt en person enligt detta sätt så kan ingen annan person tilldelas samma id inom denna domän. [OMG 1998b]

som går att tillämpa för de mest använda terminologisystemen. Målet är alltså att olika terminologisystem skall kunna erbjuda samma TQS-gränssnitt, så att applikationer som gör förfrågningar till terminologisystemen enbart behöver vara anpassade för detta gränssnitt. [OMG 1998a, Culpepper 2000]

Tjänst för kliniska observationer

Tjänsten för kliniska observationer (COAS, eng. Clinical Observation Access Service) är en standard som erbjuder en tjänst för att komma åt och göra förfrågningar på information i hälsovårdsrelaterade informationssystem. Även om namnet på standarden anspelar på klinisk information så kan COAS, förutom för klinisk information, även användas för övrig information, så som demografisk information, fakturering och administrativ information. COAS erbjuder, i likhet med TQS, ett gemensamt gränssnitt för förfrågningar till de olika system som erbjuder klinisk samt övrig information. [OMG 2000b, Culpepper 2000]

Kontroll av åtkomst till resurser

Säkerhet och konfidentiell information är något som i hög grad angår informationssystem som behandlar patientuppgifter. Detta innebär att man måste kunna kontrollera åtkomsten till information. Detta kan ske på applikationsnivå, dvs. man kontrollerar vem som har rätt att använda en applikation. Åtkomsten kan även regleras enligt de uppgifter som man vill komma åt, så att åtkomsten till valda delar av patientuppgifterna så som t.ex. testresultatet från ett HIV test är mer strikt reglerade än övriga delar av patientuppgifterna. Detta har lett till att systemleverantörerna byggt in funktionalitet för reglering av åtkomst i de system de levererar. För en hälsovårdsorganisation med många olika system i bruk leder detta till att administrationen av en säkerhets- och åtkomstpolicy som täcker hela organisationen är en mycket krävande och komplex uppgift. [Culpepper 2000]

Resource Access Decision (RAD) är en standard som definierar ett gränssnitt för auktorisering och reglering av åtkomst till information. Genom detta gränssnitt kan en applikation göra en förfrågan till ett system utanför applikationen och låta detta system sköta auktoriseringen. Målet med RAD är att en hälsovårdsorganisation skall kunna administrera en säkerhets- och åtkomstpolicy som täcker hela organisationen,

utan att behöva upprätthålla denna policy skilt i vart och ett av informationssystemen. [Culpepper 2000, OMG 1999]

Tjänst för kliniska bilder

Tjänsten för kliniska bilder (CIAS, eng. Clinical Image Access Service) är en standard som definierar ett gränssnitt genom vilket informationssystem kan utbyta kliniska bilder så som röntgenbilder. Den allmänt använda standarden för kliniska bilder är Digital Imaging and Communications in Medicine (DICOM). Denna standard inbegriper även upprätthållande av kliniska bilder så som lagring och överföring av bilder. CIAS är avsedd att användas för de fall där bildbehandlingen inte är lika omfattande som DICOM och där DICOM-kvalitet inte är nödvändig. CIAS ger möjlighet att t.ex. konvertera bilder från DICOM till mer allmänna bildformat så som GIF eller JPEG. [Culpepper 2000, OMG 2000a]

3.2.3 PlugIT

PlugIT är ett finländskt forsknings- och utvecklingsprojekt för applikationsintegrering inom hälsovården. PlugIT ägde rum åren 2001-2004 och utfördes i samarbete mellan Kuopio Universitet och yrkeshögskolan Savonia. Därtill deltog även 15 företag, 6 sjukvårdsdistrikt, en kommun och en samkommun i projektet. [Korpela 2004]

Projektets målsättning var att förbättra möjligheterna för applikationsintegrering inom hälsovården genom att ta fram öppna specifikationer på applikationsgränssnitt samt metoder och kunnande för dessa gränssnitt. Projektet producerade två typers applikationsgränssnitt: kontexthantering samt kärntjänster. [Korpela 2004]

Kontexthantering

Kontexthanteringsgränssnittet baserar sig på HL7s CCOW standard. Målet med denna specifikation är kontexthantering på miniminivå, dvs. att hitta de allra nödvändigaste delarna för att kunna bygga en grundlösning som kan hantera användar- och patientkontext. Gränssnittet är således en något modifierad och nedskalad version av CCOW. [Tuomainen 2004] Denna specifikation av kontexthantering på miniminivå blev godkänd som nationell standard av HL7 Finland i oktober 2004. [Korpela 2004]

Kärntjänster

Inom informationssystem för hälsovården finns det viss information som är gemensam för de flesta applikationer, så som patientuppgifter, användarinformation, användarrättigheter samt terminologi, här kallad kodsystem. Målet med gränssnittet för kärntjänster är att enbart en applikation, kallad kärnapplikation, skulle upprätthålla denna gemensamma information och erbjuda tillgång till denna information i form av tjänster, så kallade kärntjänster. Övriga applikationer kan alltså få tillgång till den gemensamma informationen genom de kärntjänster som finns definierade i gränssnittet. Man har definierat ett gränssnitt för användarrättighets- och patienttjänster samt ett gränssnitt för kodsystemtjänster. För dessa gränssnitt finns det både specifikationer som är oberoende av teknologi samt specifikationer som bygger på XML och HTTP. [Korpela 2004, PlugIT 2004]

3.3 Generella standarder

I de följande styckena tas de vanligaste generella standarder som används för integrering inom hälsovården upp.

3.3.1 Extensible Markup Language

Extensible Markup Language (XML) är en rekommendation från World Wide Web Consortium (W3C). XML är en förenklad version av Standard Generalized Markup Language (SGML), en standard för skapandet av en dokumentstruktur [Whatis 2004b]. XML skapades ursprungligen som ett format för textbaserade dokument, men har senare börjat användas även som dataformat samt som ett verktyg för att manipulera data [Newcomer 2002]. Med hjälp av XML kan man skapa olika dataformat, eller format för informationsutbyte, som är gemensamma för flera parter. På detta sätt kan man dela både data och dataformat mellan applikationer. [Whatis 2004b].

XML-dokument byggs upp av element som beskriver de data som dokumentet innehåller [Newcomer 2002]. I XML separeras alltså data från de dataformat som enskilda applikationer använder för att lagra data [Newcomer 2002]. Man kan jämföra XML med HTML, båda dessa format innehåller märkord som beskriver dokumentets innehåll. I HTML beskriver dessa märkord hur innehållet skall visas upp åt användaren, i XML å andra sidan beskriver märkorden hurudana data

innehållet representerar [Whatis 2004b]. Inom applikationsintegrering ger detta en lösning på problemet med olika dataformat för olika applikationer. Utan ett gemensamt dataformat är applikationerna tvungna att kunna omstrukturera sina data till vart och ett av de dataformat som de övriga applikationerna använder sig av. Detta alltså för att applikationerna skall på ett vettigt sätt kunna utbyta data. Med hjälp av XML kan man ta i bruk ett gemensamt dataformat, så att de enskilda applikationerna som utbyter information med andra applikationer via XML endast behöver kunna omstrukturera data mellan sitt interna dataformat och det gemensamma XML-formatet. [Newcomer 2002]

I samband med XML finns det ett antal relaterade tekniker och begrepp. Nedan finns en lista på de tekniker och begrepp som berör integrering:

- DTD – Document Type Definition

  I XML kan man definiera egna element som används för att beskriva innehållet. DTD är ett dokument där man kan ange vilka element som används, hurudana data dessa element innehåller samt hur dessa element är strukturerade. Man kan använda DTD för att validera dokument, dvs. kontrollera att ett dokuments struktur följer definitionen i DTD. [Whatis 2004a, Newcomer 2002]
- XML scheman

  XML scheman utvecklades som ett alternativ till DTD för att kringgå vissa begränsningar som finns i DTD. De begränsningar som XML scheman löser är bl.a. att det inte går att definiera datatyper i DTD och att DTD-definitioner är globala, dvs. elementens namn går inte att använda flera olika sammanhang för flera än en betydelse. [Newcomer 2002]
- XSL – Extensible Stylesheet Language

  XSL är ett verktyg för att omvandla ett XML-dokument till en form som kan presenteras för användare. XSLT (Extensible Stylesheet Language: Transformations) som är en del av XSL, är speciellt användningsbart inom integrering eftersom det kan användas även till att omvandla ett XML-dokument från ett XML schema till ett annat schema. [Newcomer 2002]

- XML namnrymder

  Med hjälp av XML namnrymder kan man ge unika prefix åt de olika elementen som förekommer i XML-dokument. Detta kan användas för att undvika problem som kan uppstå när samma namn på element används för flera olika definitioner på element. [Newcomer 2002]

3.3.2 Elektronisk överföring av strukturerade data

Elektronisk överföring av strukturerade data (EDI, eng. Electronic Data Interchange) är en metod för att utbyta data mellan informationssystem. Ursprungligen användes EDI för att överföra beställningar och fakturor mellan företag eller offentliga organisationer. Med tiden har det utvecklats för att överföra även annan information mellan organisationer [Stultz 2001]. EDI är inte i sig själv en standard, utan det finns flera standarder för EDI [Stultz 2001]. Enligt Stultz (2001) är dessa de ledande EDI-standarderna: EDI For Administration, Commerce, and Transport (EDIFACT) och American National Standards Institute/Accredited Standards Committee (ANSI/ASC) X12. Av dessa används EDIFACT i Finland och det övriga Europa [Savolainen 2004, Stultz 2001]. EDIFACT-standarden innehåller definitioner på meddelanden genom vilka utbytet av information sker, bland dessa definitioner finns det även meddelanden avsedda för att användas inom hälsovården. De meddelanden som används för hälsovården är [Kyöste 2002]:

- begäran av undersökning samt undersökningsresultat
- remiss
- patologiskt utlåtande
- EDI-meddelande för läkarutlåtande (detta meddelande är ännu under utveckling)

3.4 Överblick av läget i Finland

Enligt en kartläggning av informationssystemen inom hälsovården år 2001 [Hautsalo 2002] så var HL7 den mest använda standarden för integrering inom hälsovården i Finland år 2001. EDIFACT var en annan betydande standard, trenden är dock den att HL7 ökar i popularitet medan användningen av EDIFACT minskar. Samma undersökning nämner XML som en växande standard för meddelanden inom integreringen. [Hautsalo 2002]

Kärkkäinen [Kärkkäinen 2001] nämner samma trend i sin undersökning år 2001, dvs. att användningen av HL7 ökar medan användningen av EDIFACT minskar. En fördel med HL7 framom EDIFACT är att HL7 används såväl till intern dataöverföring inom organisationen som till dataöverföring organisationer emellan. EDIFACT används i regel enbart för dataöverföring organisationer emellan [Kärkkäinen 2001]. Kärkkäinen nämner också XML som en standard som kommer att växa i framtiden. [Kärkkäinen 2001].

PlugIT-projektets standarder är från år 2004 och således finns det inte i skrivande stund information om i hurudan omfattning dessa standarder tagits i bruk. Flera större finska systemleverantörer deltog dock i projektet och man kan vänta sig att dessa så småningom tar i bruk de standarder som projektet producerade. [Korpela 2004]

4 Kravspecifikation

4.1 Utgångsläge

Denna kravspecifikation utgår ifrån att man på en sjukvårdsanstalt använder RAIsoft-LTC. Parallellt med detta så används även ett patientadministrationssystem och eventuella övriga specialsystem. I utgångsläget är dock dessa system, både patientadministrationssystemet och de övriga systemen, helt fristående i förhållande till RAIsoft-LTC.

Detta scenario, där man har flera olika separata applikationer i användning på samma sjukvårdsanstalt, innebär att läkaren eller vårdpersonalen använder flera program samtidigt när man behandlar patientens uppgifter. Vid en läkarmottagning kan användaren ha upp till fyra olika program i användning samtidigt för att kunna få fram eller spara alla de patientuppgifter som är nödvändigt [Hautsalo 2002]. I ett dylikt scenario förekommer det vissa problem:

- Användaren utför repetitivt arbete, som att skriva in patientuppgifter i flera applikationer. Ofta är det fråga om samma information som skrivs in de olika applikationerna.
- När användaren uppdaterar patientinformation så måste användaren byta mellan flera olika applikationer. Detta innebär att det går åt tid till inloggningen och till att söka fram patientens uppgifter i vart och ett av applikationerna.
- Det föreligger en risk att informationen är inkongruent, dvs. att informationen om en patient i en applikation inte stämmer överens med motsvarande information i en annan applikation.

I en dylik omgivning kan det också uppstå problem när patienten byter vårdplats, internt eller mellan sjukvårdsanstalt. Man måste samla patientens information från flera applikationer, kopiera informationen och föra över den till de applikationer som är i bruk vid den nya vårdplatsen. Ofta sker detta manuellt, vilket innebär att det finns en risk för att all information inte överförs korrekt. Om information sedan saknas eller är felaktig på den nya vårdplatsen så kan man vara tvungen att utföra

samma undersökningar på patienten som redan en gång gjorts på den förra vårdplatsen [Hautsalo 2002].

Man kan dra den slutsatsen att bristen på kommunikation mellan applikationerna leder till att dessa applikationer tar upp onödiga resurser av vårdpersonalen. Ifall man kunde minimera de resurser som vårdpersonalen använder till att göra repetitivt arbete med flera olika applikationer, så kunde dessa resurser användas till att vårda patienten eller utföra annat viktigt arbete som har direkt med vården att göra.

Målsättningen med denna kravspecifikation är att ställa upp krav för ett integreringsgränssnitt för RAIsoft-LTC som bemöter dessa problem och som kan effektivera och stöda användarens arbete.

4.2 Intressegrupper

När det gäller att planera en integreringslösning finns det ett antal intressegrupper som har olika intressen i denna lösning. Nedan är en lista på de viktigaste intressegrupperna samt deras intressen.

4.2.1 Systemleverantören – Oy Raisoft Ltd

Systemleverantören är företaget som utvecklar och säljer RAIsoft-LTC. I systemleverantörens intressen ligger följande:

- Kundens behov – det är viktigt att bemöta kundens behov och utveckla en produkt som passar för kundens behov.
- Produktutveckling/teknisk lösning – det är viktigt att utveckla en generell integreringslösning som går att återanvända i flera produkter och med flera olika systemleverantörer. Det är också viktigt att lösningen går att återanvända i det hänseendet att man kan utöka den men nya funktionaliteter för kommande behov.
- Systemleverantören måste komma överens med de övriga systemleverantörerna om vilken teknik som används för utbyte av information mellan systemleverantörens egen programvara och övriga programvaror. Här ligger systemleverantörens intresse i en lösning som passar bra med övriga tekniska lösningar i den egna produkten.
- Ekonomiska intressen – det är viktigt att den lösning som utvecklas är ekonomiskt lönsam och betalar sig själv i någon form tillbaka.

Ur systemleverantörens synvinkel finns det alltså ett behov av att utveckla en generell teknisk lösning som löser kundens problem och som går att återanvända i flera produkter. För att den tekniska lösningen ska gå att använda i samarbete med flera olika systemleverantörer så bör den bygga på en allmänt använd teknologi. I mån av möjlighet bör man använda sig av tekniska standarder eller *de facto* standarder som även andra systemleverantörer stöder i sina produkter. På så sätt kan man underlätta processen för att integrera den egna produkten med en annan systemleverantörs produkt.

4.2.2 Användare – vårdpersonal

Vårdpersonal på sjukvårdsanstalter är i regel de som använder RAIsoft-LTC. Det primära intresset för de organisationer där applikationen används är att kunna ge patienterna så bra vård som möjligt. Ur detta kan man med tanke på systemintegrering härleda följande intressen:

- Minimering av onödigt arbete – minimera de resurser som sätts på att utföra repetitivt arbete, så som att kopiera och flytta information mellan de olika applikationerna.
- Inga stora synliga förändringar i användargränssnittet – ju mera användargränssnittet ändrar från det användaren är van vid, desto mera tid går det åt att lära sig använda det nya användargränssnittet.
- Integreringslösningen stöder vårdprocessen.

Ur användarens synvinkel finns det alltså vissa scenarion som man vill effektivera genom integrering. Det gäller för det första situationer där det sker en händelse som medför att användaren måste skriva in samma information i flera applikationer, t.ex. patientadministrationssystemet och RAIsoft-LTC. Exempel på detta är inskrivning av patient, beskrivet nedan i användningsfall 1, och utskrivning av patient. För det andra gäller det situationer där användaren är tvungen att samtidigt använda flera applikationer för att få tillgång till och eventuellt kunna göra ändringar i alla de uppgifter som berör patienten. Exempel på detta är när man vill utforma en rapport om patientens tillstånd, beskrivet nedan i användningsfall 2, och planering av vården.

Användningsfall 1. Inskrivning av patient

Användningen av applikationerna vid inskrivning av en patient ser ut som följande:

1. Användaren öppnar patientadministrationssystemet.
2. Användaren skriver in patientens uppgifter i patientadministrationssystemet.
3. Användaren loggar in i RAIsoft-LTC (detta kan ske i samband med att användaren skriver in patienten i patientadministrationssystemet, eller i ett senare skede då man vill använda RAIsoft-LTC för att göra en bedömning på patienten).
4. Användaren skriver in patientens uppgifter i RAIsoft-LTC.

Problemet, som skall effektiveras i detta användningsfall, är att användaren måste fylla i samma information i de båda olika applikationerna. Dvs. användaren gör samma sak flera gånger, det borde räcka med att fylla i informationen endast en gång. På så sätt kunde användaren spara tid och undvika att t.ex. i misstag fylla i olika uppgifter i de olika applikationerna, så att informationen om patienten i en applikation är inkongruent med informationen om samma patient i en annan applikation.

Detta problem kan man lösa genom att patientadministrationssystemet uppdaterar uppgifterna i RAIsoft-LTC automatiskt när användaren har skrivit in patientens uppgifter i patientadministrationssystemet. Alternativt kan RAIsoft-LTC kolla patientuppgifterna från patientadministrationssystemet när användaren aktiverar patienten i RAIsoft-LTC.

Användningsfall 2. Utformning av rapport om patientens tillstånd

Användningen av applikationer vid utformning av rapport om patientens tillstånd ser ut som följande:

1. Användaren loggar in i patientadministrationssystemet och söker fram patientens uppgifter och skriver ut dem.

2. Användaren loggar in i RAIsoft-LTC och söker fram patientens uppgifter och skriver ut dem.

Problemet, som skall effektiveras i detta användningsfall, är att användaren måste logga in och söka fram patientens uppgifter ur flera olika applikationer. Det borde räcka med att användaren endast på en applikation söker fram patientens uppgifter. På så sätt kunde användaren spara tid och automatiskt få fram all information som finns att fås angående patienten.

Detta problem kan lösas på två olika nivåer: dels genom kontexthantering [Tuomainen 2004] och dels genom att patientadministrationssystemet begär den behövliga informationen av RAIsoft-LTC. Kontexthantering innebär att de olika programmen som användaren använder får information om vilken patients uppgifter som behandlas för tillfället. Se kapitel 3 för mera information om kontexthantering.

4.2.3 Kunderna

Kunderna som köper RAIsoft-LTC är i regel hälsovårdscentraler, åldringshem och vårdhem. I kundernas intressen ligger samma intressen som användarnas intressen. Förutom detta så har kunderna även ekonomiska intressen: kunden vill betala endast för det mervärde som systemintegreringen ger åt kunden. Detta mervärde kan ta sig i uttryck på flera olika sätt:

- Tid – Ifall mindre tid går åt att utföra repetitivt arbete med applikationerna får personalen mer tid för vårdarbetet. Detta alltså utan att man förlorar något i nyttan med applikationerna.
- Vårdkvalitet – Vårdkvaliteten förbättras i och med att vårdpersonalen får mera tid för patienten. Vårdkvaliteten förbättras också i och med att informationsutbytet mellan olika applikationer sker snabbare och mer felfritt än i de fall där man manuellt för över informationen från en applikation till en annan. Detta gäller i synnerhet situationer där patienten flyttas mellan olika vårdplatser.

4.2.4 IT-ansvariga

IT-ansvariga är de som sköter om att ta i bruk samt att administrera programvaran. Dessa kan vara kommunala IT-ansvariga, privata organisationers IT-ansvariga eller konsulter. I viss mån sköter även systemleverantören om ibruktagandet av

programvaran. Ur de IT-ansvarigas synvinkel är det viktigt att programvaran är lätt att ta i bruk och lätt att upprätthålla. Det är också i de IT-ansvarigas intresse att programvaran använder sådan teknik som är allmän i dylika programvaror och inte kräver stora ändringar i IT-systemen utöver programvaran själv.

4.2.5 Övriga systemleverantörer

Övriga systemleverantörer är företag som utvecklar och säljer de övriga applikationer som används vid sidan om RAIsoft-LTC. I dessa systemleverantörers intressen ligger följande:

- Ekonomiska intressen – det är viktigt att den lösning som utvecklas är ekonomiskt lönsam och betalar sig själv tillbaka.
- Valet av teknisk lösning för utbyte av information. Systemleverantörerna har intresse i valet av teknik för att utbyta information mellan applikationer, så att den passar bra ihop med systemleverantörens egen applikations övriga tekniska lösningar.

4.2.6 Patienten och myndigheter

När det gäller applikationer som behandlar personliga uppgifter om patienter så får inte patientens integritetsskydd brytas. Detta innebär att patienten måste skriftligen ge samtycke till utlämning av uppgifter till utomstående [Finlex 2004]. Följaktligen får inte patientinformation utlämnas och överföras till applikationer eller användare av applikationer som kan klassas som utomstående. En situation som beskriver detta är när patienten byter vårdplats och man vill överföra patientens vårdinformation från den förra vårdplatsen till den nya vårdplatsen.

4.3 Krav

För att den planerade integreringslösningen skall uppfylla de ovannämnda behoven så kan man ställa följande krav på integreringen:

1. **Utbyte av information mellan RAIsoft-LTC och övriga applikationer**
   1.1. RAIsoft-LTC måste kunna ta emot och ge ut patienters personuppgifter, detta bör ske vid patientens inskrivning eller när det sker ändringar i patientens personuppgifter. När RAIsoft-LTC tar emot personuppgifter så

bör dessa uppdateras så att patientens personuppgifter är korrekta i alla applikationer.

1.2. RAIsoft-LTC måste kunna ge ut klinisk information om patienten till övriga applikationer. Detta bör ske så att den utomstående applikationen kan begära informationen av RAIsoft-LTC.

1.3. RAIsoft-LTC måste kunna ta emot information om förändringar i patientens vårdsituation och uppdatera dessa förändringar i den egna databasen. Förändringar i vårdsituationen kan vara t.ex. inskrivning, utskrivning, byte av avdelning.

1.4. RAIsoft-LTC måste kunna startas upp automatiskt på användarens arbetsstation och kunna ta emot kontextinformation om vilken patient som skall visas upp åt användaren.

2. Säkerhet

2.1. Tillgång till patientinformation måste kunna begränsas så att endast användare som har rätt att använda en patients uppgifter får tillgång till dessa uppgifter.

2.2. Vid automatiskt utbyte av patientrelaterad information mellan applikationer bör applikationerna ta i beaktande ifall patienten har givit sitt samtycke för utlämnande av dessa uppgifter.

2.3. När information överförs mellan applikationer bör användarrättigheterna till denna information tas i beaktande, så att informationen är skyddad enligt samma policy i den applikation som mottar informationen som i den applikation som ger ut informationen. Dvs. när man överför konfidentiell information från en applikation till en annan så bör informationen klassas som konfidentiell på samma nivå i den andra applikationen.

2.4. Informationen och informationsutbytet mellan applikationer måste skyddas från insyn av eventuella inkräktare i systemet. Detta behöver inte nödvändigtvis vara inbyggt i integreringslösningen, utan kan också vara en del av den omgivning där applikationerna är i bruk.

3. Teknologi

3.1. Kommunikationen mellan RAIsoft-LTC och de övriga applikationerna bör så långt som möjligt ske genom ett gränssnitt som kan användas för flera

lösningar och som inte behöver skräddarsys skilt för varje lösning. I detta gränssnitt bör man om möjligt använda sig av tekniker som kan klassas som standarder eller *de facto* standarder för att underlätta återanvändningen av gränssnittet och för att underlätta samarbetet med övriga systemleverantörer.

3.2. Även om lösningen bör vara återanvändbar så är det dock inte ändamålsenligt, med tanke på de ekonomiska kraven, att utveckla en lösning som löser alla problem och som går att använda i alla potentiella situationer. Målsättningen är snarare att skapa en grundlösning för de behov som finns i dagens läge, på ett sådant sätt att denna lösning kan vidareutvecklas när det uppkommer nya behov.

4. Ekonomi

4.1. Den grundlösning som här planeras bör kunna tas i bruk och säljas som en del av Raisofts produkter. Dvs. man vill utveckla en lösning som svarar på de behov som kunderna har i det nuvarande läget, man vill inte lösa sådana problem som kunden inte ännu har.

4.4 Sammanfattning

De krav som ställts i denna kravspecifikation gäller egenskaper hos det planerade integreringsgränssnittet. Dvs. detta är vad man vill åstadkomma med denna lösning, samt tekniska och ekonomiska ramar för lösningen. Krav på hur denna lösning skall rent tekniskt fungera ställs inte här, de hör till den tekniska specifikationen i det femte kapitlet.

5 Teknisk specifikation

5.1 Systemarkitektur

5.1.1 Nuvarande systemarkitektur

RAIsoft-LTC bygger på en tvåskikts klient-server arkitektur med en så kallad tjock klient. Tvåskikts klient-server system består av tre delar: databas, applikationslogik och användargränssnitt. Databasen finns på en server, användargränssnittet finns på klienten, normalt på användarens arbetsstation, och applikationslogiken kan vara implementerad på servern, på klienten eller uppdelad på både servern och klienten. I en tjock klient är applikationslogiken implementerad helt på klienten tillsammans med användargränssnittet. [Sadoski 2004] RAIsoft-LTC bygger alltså på denna arkitektur och i denna lösning är användargränssnittet och applikationslogiken tätt sammankopplade, klientens programvara består i stort sett av en enda exekverbar fil. Bild 5 beskriver denna uppläggning av systemet.

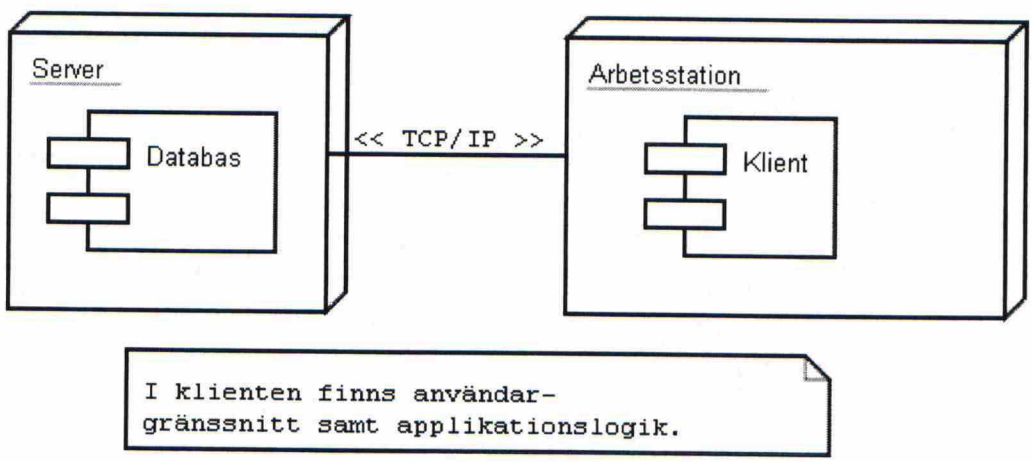

Bild 5 Systemarkitektur i det nuvarande systemet.

5.1.2 Systemarkitektur som stöder integreringsgränssnitt

När det gäller de krav på integreringsgränssnittet som ställdes i kravspecifikationen så är man intresserad av att kunna integrera databas, applikationslogik och användargränssnitt med andra applikationer. Den tekniska lösningen skall också vara återanvändbar, dock inte så att man löser alla potentiella problem på en gång.

Återanvändbarheten bör tillämpas på ett sätt som möjliggör att den tekniska lösningen lätt kan vidareutvecklas. Återanvändbarheten betyder även att man bör kunna ta produkten i bruk i olika tekniska omgivningar. Det innebär att det bör vara enkelt att lägga till stöd för nya tekniska omgivningar eller teknologier, som dittills inte varit aktuella inom integreringen.

På basen av dessa krav är det vettigt att inom integreringen separera det som är beroende av teknologi ifrån det som inte är teknologiberoende. Överlag är de funktionaliteter som integreras inte beroende av den teknologi som används inom integreringen. Det är snarare integreringsgränssnittet som erbjuds åt andra applikationer som är beroende av den integreringsteknologi som används. Man bör alltså använda sig av ett integreringsgränssnitt som kan komma åt de funktionaliteter i applikationen som skall integreras och kan översätta dessa till den teknologi som används för applikationsintegreringen. Man kunde även kalla detta integreringsgränssnitt för en adapter mellan RAIsoft-LTC och de övriga applikationerna [Linthicum 2004]. Ifall man bygger upp detta gränssnitt som en separat komponent ifrån den funktionalitet man vill integrera, så kan man lätt utöka gränssnittet så att det stöder fler integreringsteknologier i efterhand utan att behöva ändra på den del av applikationen där funktionaliteten är implementerad.

Det finns enligt kravspecifikationen behov av att kunna integrera vissa funktionaliteter av applikationen som finns i databas och applikationslogik utan att användargränssnittet berörs. Detta behov talar för att man borde separera dessa funktionaliteter från klientens användargränssnitt så att övriga applikationer kan komma åt dessa funktionaliteter utan att behöva aktivera användargränssnittet. Bild 6 beskriver detta.

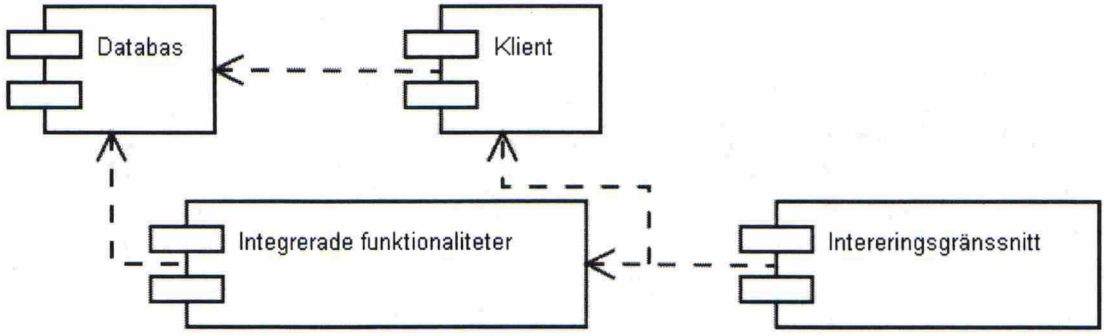

Bild 6 De funktionaliteter som skall integreras är separerade från klientens användargränssnitt.

Ett exempel på en situation som kan undvikas genom detta är när man på en arbetsstation kör en applikation som integrerats med RAIsoft-LTC, men RAIsoft-LTCs klientprogramvara i sig inte är i bruk på denna arbetsstation. Dvs. man använder inte RAIsoft-LTCs användargränssnitt på denna arbetsstation. Ifall de funktionaliteter som används genom integrering inte är separerade från användargränssnittet är man i detta fall tvungen att installera RAIsoft-LTCs klientprogramvara, som även innehåller användargränssnittet, endast för att komma åt funktionaliteterna som berör applikationslogiken.

Det kräver mycket resurser att separera de funktionaliteter i applikationslogiken som används inom integreringen från användargränssnittet, eftersom det innebär att man måste göra ändringar i klientprogramvarans struktur. Detta innebär i sin tur att man måste utföra test på hela applikationen för att kontrollera att dessa ändringar inte medfört några fel i applikationen. Dessa test medför en stor kostnad varje gång man vill integrera en ny funktionalitet med övriga applikationer, eftersom man varje gång måste testa klientprogramvaran.

Ett vettigare tillvägagångssätt, med tanke på återanvändbarhet och möjligheter för vidareutveckling, är att separera alla funktionaliteter inom applikationslogiken från användargränssnittet, vare sig dessa funktionaliteter används inom integreringen eller inte. Denna uppdelning bör ske på ett sådant sätt att man lätt kan använda applikationslogiken fristående från användargränssnittet. Detta innebär ett mer omfattande arbete för att strukturera om programvaran, men nyttan av detta får man i framtiden då det uppkommer nya behov för att integrera funktionaliteter i applikationslogiken som inte integrerats tidigare. I dessa fall finns funktionaliteten

färdigt separat från användargränssnittet och kostnaden för att integrera den nya funktionaliteten blir betydligt mindre. Detta tillvägagångssätt stöder det kravet i kravspecifikationen som säger att den tekniska lösning som här beskrivs ska vara sådan att den enkelt går att vidareutveckla när det uppkommer nya behov för integrering. Bild 7 beskriver denna uppläggning av systemet.

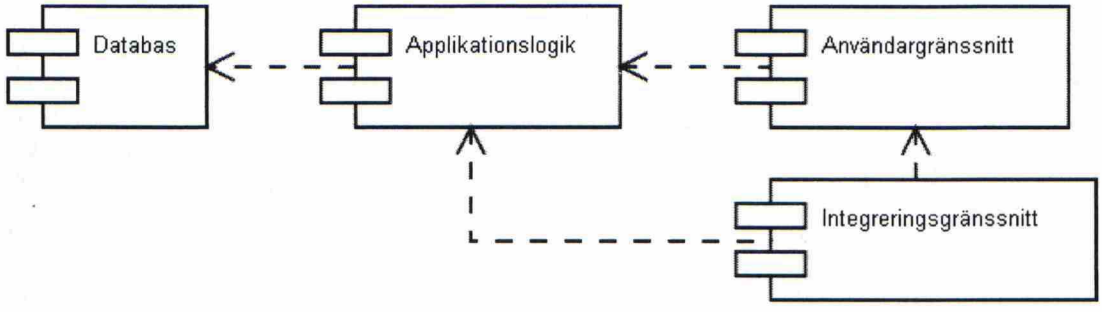

Bild 7 All applikationslogik separat ifrån användargränssnittet.

För att kunna använda funktionaliteterna både för klientens programvara samt inom applikationsintegreringen är det ändamålsenligt att bygga upp programbibliotek som innehåller all den funktionalitet av applikationen som går att spara i generella bibliotek. Dessutom bör man använda sig av väldefinierade gränssnitt genom vilka applikationen använder dessa programbibliotek. Målet är att den programkod som överför funktionaliteten till integreringsgränssnittet kan använda sig av de samma gränssnitten som klientens programvara använder sig av.

Problemet med denna systemarkitektur, där all applikationslogik är separerad från användargränssnittet, är att den kostnad som omstruktureringen för med sig inte betalar sig tillbaka med det samma. Denna modell strider till en viss del mot kraven 3.2 och 4.1 i kravspecifikationen som säger att det varken är ändamålsenligt eller ekonomiskt att utveckla en lösning som löser sådana problem som inte kunden har. Å andra sidan stöder denna systemarkitektur de kraven i del 3 av kravspecifikationen som säger att lösningen bör vara återanvändningsbar och vara lätt att vidareutveckla. Man borde alltså göra en kompromiss mellan dessa två målsättningar, 1) ändamålsenlighet med tanke på kundernas nuvarande behov och kostnader och 2) återanvändbarhet.

En kompromiss som ligger mellan dessa målsättningar ser ut som följande:

- Man separerar från klientens programvara endast de funktionaliteter som behövs i det nuvarande läget för integrering.
- Framöver när programvaran utvecklas och nya funktionaliteter läggs till så lägger man till dessa funktionaliteter enligt modellen för återanvändning, dvs. separat från användargränssnittet.

5.2 Modell för integreringsgränssnitt

På basen av kravspecifikationen och systemarkitekturen så kan man definiera ett integreringsgränssnitt som RAIsoft-LTC kan erbjuda övriga applikationer för integreringssyften. Integreringsgränssnittet delas upp i två skikt: ett API-skikt, och ett adapterskikt. Tanken är att API-skiktet använder sig av funktionaliteter av applikationslogiken, vid behov även av användargränssnittet, och erbjuder dessa funktionaliteter som ett applikationsgränssnitt åt övriga applikationer. Adapterskiktet översätter de funktioner som erbjuds i API-skiktet till en viss mellanprogramsteknologi genom vilken övriga applikationer kan använda integreringsgränssnittet, t.ex. CORBA, Web Services, etc.

De funktionaliteter som integreringsgränssnittet behöver använda sig av delas här in i tre olika moduler: autentisering, patientinformation och kontexthantering. Dessa moduler är beskrivna i de följande styckena.

Modulen för autentisering

Integreringsgränssnittet bör, för att fylla säkerhetskraven i kravspecifikationen, ha en möjlighet att autentisera användare som via övriga applikationer använder integreringsgränssnittet. Denna modul kontrollerar att användaren har rätt att använda de uppgifter som hämtas eller uppdateras via integreringsgränssnittet samt loggar in användaren till databasen ifall autentiseringen är korrekt. Autentiseringen kan ske enligt samma principer som RAIsoft-LTC följer då användaren loggar in i programmet via användargränssnittet, alternativt kan denna modul använda sig av en autentiseringstjänst som någon annan applikation erbjuder, t.ex. OMGs RAD [OMG 1999]. Denna modul behöver inte nödvändigtvis vara synlig i integreringsgränssnittet som separata funktioner, utan kan även vara en del av de övriga funktioner som integreringsgränssnittet erbjuder.

Eftersom detta är en modul som inte är beroende av användargränssnittet så borde funktionaliteten för autentisering separeras från klientens programvara till en självständig modul eller ett programbibliotek som både klientens programvara och integreringsgränssnittet kan använda sig av. Detta befrämjar både återanvändbarhet samt underhåll av programkoden.

För autentiseringsmodulen kan man definiera följande funktion:
authenticateUser(username, password, patient) : AuthenticationResult
Denna funktion tar alltså användarnamn, lösenord och patientidentifierare som argument och loggar in användaren ifall användaren har tillgång till patientens uppgifter. Returvärdet beskriver hur autentiseringen lyckades.

Modulen för patientinformation

Det bör finnas en modul för patientinformation för att fylla de krav på utbyte av information som ställs i kravspecifikationen. För att erbjuda den funktionalitet som denna modul erbjuder krävs det att API-skiktet använder sig även av autentiseringsmodulen, för att förhindra obehöriga användare från att läsa eller uppdatera patientinformation som de inte har användarrätt till.

Eftersom detta är en modul som inte är beroende av användargränssnittet så borde funktionaliteten som beskrivs nedan separeras från klientens programvara till en självständig modul eller ett programbibliotek som både klientprogramvaran och integreringsgränssnittet kan använda sig av. Detta befrämjar både återanvändbarhet samt underhåll av programkoden.

För att kunna få tillgång till patientuppgifter så måste det finnas ett sätt att identifiera patienten på. Inom ramen för denna specifikation är patientens socialskyddssignum ett tillräckligt tillförlitligt sätt att identifiera patienter på. Nämnas bör dock att det finns situationer som denna identifieringsmetod inte täcker: t.ex. vid behandling av utländska patienter kan det förekomma fall då patienten är utan inhemskt socialskyddssignum.

Den patientinformation som applikationer kan få tillgång till via integreringsgränssnittet kan delas in i två delområden:

1. **Uppgifter som är gemensamma för flera applikationer – patientens personuppgifter**

RAIsoft-LTC skall kunna ta emot information om patienters vårdsituation[3] samt patienters personuppgifter, så som namn, födelsedatum och hemkommun. Detta behov uppkommer vid inskrivning av en ny patient samt då patientens personuppgifter förändrats. Detta kan ske på två sätt:

- RAIsoft-LTC hämtar uppgifterna från en annan applikation, detta bör ske automatiskt vid inskrivning av patient. I de fall då patientuppgifter uppdaterats i en annan applikation så måste det finnas en mekanism för RAIsoft-LTC att veta när patientuppgifterna ändrats och bör hämtas.
- Övriga applikationer, där patientuppgifter skrivs in, sänder automatiskt informationen till RAIsoft-LTC via integreringsgränssnittet.

Det kan också uppkomma behov för att RAIsoft-LTC ska ge ut patienters personuppgifter, i dessa fall bör integreringsgränssnittet erbjuda en mekanism för övriga applikationer att göra förfrågningar om patienters personuppgifter. RAIsoft-LTC bör även meddela ett eventuellt patientadministrationssystem om patientens personuppgifter uppdaterats.

2. **Uppgifter som är specifika för RAIsoft-LTC – klinisk information om patienten**

Integreringsgränssnittet bör enligt kravspecifikationen ge ut klinisk information[4] om patienter. Detta gäller alltså sådan information som är specifik för RAIsoft-LTC och inte genereras i andra applikationer. Det finns alltså ett behov för applikationer att kunna göra förfrågningar på denna information via integreringsgränssnittet. Dessa applikationer behöver dock inte uppdatera denna typ av information via integreringsgränssnittet eftersom informationen genereras endast i RAIsoft-LTC.

---

[3] Med vårdsituation menas i detta sammanhang uppgifter om inskrivning och utskrivning.

[4] Med klinisk information menas i detta sammanhang information som beskriver det tillstånd som patienten befinner sig.

För patientinformationsmodulen kan man definiera följande funktioner:

- **updateCommonPI(patient, updateInformation)**
  Med denna funktion kan övriga applikationer ge uppdateringar i patientinformationen till RAIsoft-LTC. Parametrar är patientidentifierare och den information som uppdaterats.
- **patientInformationChanged(patient)**
  Denna funktion meddelar RAIsoft-LTC att patientinformationen ändrats och att programmet bör hämta den uppdaterade informationen. Funktionen tar patientidentifierare som parameter.
- **getCommonPI(patient, informationRequest) : CommonPI**
  Med denna funktion kan övriga applikationer hämta patientinformation som är gemensam för flera applikationer från RAIsoft-LTC. Parametrar är patientidentifierare och förfrågan om information. I den andra parametern specificeras vilken information som skall hämtas. Returvärde är den begärda informationen.
- **getRaisoftPI(patient, informationRequest) : RaisoftPI**
  Med denna funktion kan övriga applikationer hämta patientinformation som är specifik för RAIsoft-LTC. Parametrar är patientidentifierare och förfrågan om information. I den andra parametern specificeras vilken information som skall hämtas. Returvärde är den begärda informationen.

Modulen för kontexthantering

Ur kravspecifikationen framgår det ett behov av att kunna överföra information om användar- och patientkontext till RAIsoft-LTC från övriga applikationer. Detta innebär att övriga applikationer bör genom integreringsgränssnittet kunna meddela RAIsoft-LTC om förändrad användar- och patientkontext. På motsvarande sätt bör även RAIsoft-LTC kunna meddela övriga applikationer om ändringar i kontexten. Övriga applikationer bör även kunna starta upp RAIsoft-LTC och ge det en viss kontext.

Eftersom funktionaliteten i denna modul är beroende av användargränssnittet så kan funktionaliteten inte separeras från användargränssnittet på samma sätt som i de

övriga modulerna. Istället måste integreringsgränssnittet anropa funktionaliteten för denna modul direkt från klientens programvara.

I kontexthanteringsmodulen kan man specificera följande funktion:

contextChanged(userContext, patientContext)

Med denna funktion kan övriga applikationer meddela RAIsoft-LTC om att kontexten ändrats. Parametern userContext beskriver den aktiva användaren och parametern patientContext beskriver den valda patienten.

API-skiktet

Detta skikt använder sig av de tre ovannämnda modulerna för att utföra de funktionaliteter som krävs för integreringen. Utåt, åt övriga applikationer, erbjuder detta skikt ett applikationsgränssnitt som kan användas för integrering. Skiktet kan t.ex. uppfylla en viss meddelandestandard och översätta den kommunikation som sker enligt den standarden till funktionsanrop i de moduler som nämns i styckena ovan. Detta stöder återanvändning av de ovannämnda modulerna på så vis att man kan erbjuda flera olika applikationsgränssnitt, t.ex. ett meddelandeorienterat och ett RPC-orienterat applikationsgränssnitt, utan att behöva implementera själva funktionaliteten på flera olika ställen.

Adapterskiktet

Detta skikt översätter API-skiktets funktioner till en viss mellanprogramsteknologi så som CORBA, DCOM, Web Services etc. Med hjälp av detta skikt kan man hålla den programkod som bygger upp funktionaliteter samt API skild från mellanprogramsteknologier.

Klassdiagram av integreringsgränssnittet

Bild 8 visar ett klassdiagram som beskriver integreringsgränssnittet. Klasserna AuthenticationModule, PatientInformationModule och ContextModule innehåller de funktionaliteter som skall användas inom integreringsgränssnittet. De funktioner som beskrivits i styckena ovan och som används i klassdiagrammet är riktgivande och beskriver hurudana funktioner som behövs. När modellen implementeras så kan man ge funktionerna mer beskrivande namn efter behov. Det som i modellen beskrivs av en funktion kan i implementationen bestå av flera funktioner. Klassen API innehåller

API-skiktet, denna klass erbjuder som ett exempel en funktion för att göra en förfrågan. Klasserna CORBA samt WebService är exempel på adapterskiktet. Dessa klasser översätter funktionerna i API-klassen till mellanprogramsteknologier.

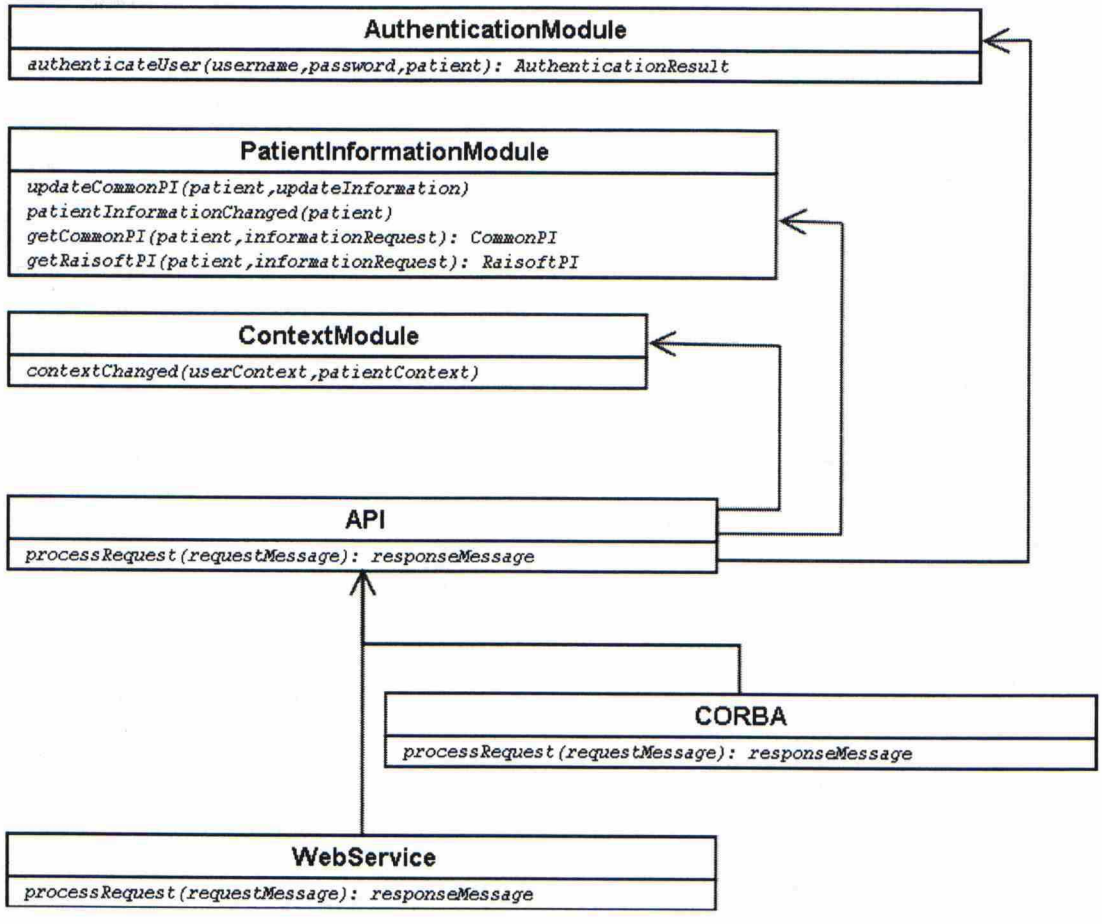

Bild 8 Klassdiagram över integreringsgränssnittet.

5.3 Integreringsstandarder

I kravspecifikationen ställs kravet på att den integreringslösning som tas i bruk bör i mån av möjlighet byggas på standarder, officiella eller de facto standarder, för att underlätta återanvändning av integreringsgränssnittet. Detta innebär att det integreringsgränssnitt som byggs för RAIsoft-LTC bör i mån av möjlighet följa en standard som även övriga systemleverantörer använder sig av. Eftersom Raisoft har ett internationellt kundunderlag så är internationella standarder att föredra framom nationella standarder.

Som det redan nämndes i kapitel 3 så är HL7 den mest använda integreringsstandarden i Finland. HL7 är också en av de viktigaste standarderna internationellt sett. Det nämndes också att XML som meddelandeformat är en växande standard inom hälsovården, medan EDIFACT är på nedgång inom denna sektor. PlugIT-projektet har givit ut nya standarder som har potential att bli väl använda i Finland (se kapitel 3).

Användandet av en integreringsstandard förutsätter att det finns även övriga applikationer som stöder samma standard i den omgivning där integreringen utförs. Eftersom denna specifikation utgör en allmän modell för integrering och inte en slutlig specifikation för en enskild integreringslösning så specificeras inte här användandet av någon viss standard. Det slutliga valet av integreringsstandard beror inte enbart på systemleverantörens egna preferenser utan även på kundens behov samt på övriga systemleverantörer. Inom ramen för denna specifikation kan dock användning av HL7 samt XML rekommenderas. Dessa är internationella standarder som är använda i stor grad och vars användning ökar för tillfället. Detta faktum betyder att möjligheterna ökar för att återanvända integreringslösningar baserade på dessa standarder såväl i Finland som internationellt. Ifall PlugIT-standarderna blir allmänna kan även dessa rekommenderas i framtiden för integreringslösningar i Finland. Dessa nämnda standarder, HL7, XML samt PlugIT, stöder de integreringsbehov som presenterats i kravspecifikationen och i denna specifikation och lämpar sig således bra för RAIsoft-LTC.

5.4 Sammanfattning

I detta kapitel har en återanvändningsbar integreringsmodell för RAIsoft-LTC utvecklats. Enligt denna modell föreslås en ändring i systemarkitekturen så att man separerar de funktionaliteter som skall integreras från klientprogramvaran i RAIsoft-LTC. Dessa funktionaliteter läggs till ett programbibliotek som kan användas både av själva klientprogramvaran och av integreringsgränssnittet. När nya funktionaliteter läggs till i applikationen föreslås att dessa läggs direkt till samma programbibliotek så att man lätt kan integrera dessa funktionaliteter vid behov.

Integreringsgränssnittet som använder sig av detta programbibliotek kan delas in i två skikt, API-skiktet och adapterskiktet. Denna uppdelning separerar de delar av integreringsgränssnittet som är beroende av mellanprogramsteknologi från de delar som inte är beroende av mellanprogramsteknologi. På så sätt är modellen mera återanvändningsbar i de fall där man t.ex. vill lägga till stöd för en ny mellanprogramsteknologi.

6 Försökstillämpning: integreringsgränssnitt för RAIsoft-LTC

För att utvärdera den modell som beskrivits i kapitel 5, så har jag gjort en försökstillämpning där jag tillämpar en del av modellen. Målet med denna försökstillämpning är att verifiera att modellen fungerar i praktiken samt att identifiera de fördelar och de problem som kommer fram när modellen implementeras.

6.1 Beskrivning av försökstillämpningen

Försökstillämpningen består av ett integreringsgränssnitt till RAIsoft-LTC samt ett litet klientprogram som använder sig av detta gränssnitt. Genom gränssnittet kan klientprogrammet göra förfrågningar på tre olika typer av patienters skalor[5]: Case-Mix Index (CMI), Cognitive Performance Scale (CPS) och Activities of Daily Living (ADL). Med klientprogrammet som kallar på integreringsgränssnittet kan man göra förfrågningar på olika kombinationer av dessa skalor. Förfrågningen sker så att klientprogrammet ger integreringsgränssnittet användarnamn, lösenord, patientens socialskyddssignum samt information om vilka skalor användaren vill se. Som resultat på förfrågningen ges patientens socialskyddssignum samt värdena på de skalor som hämtats.

Dataöverföringen i integreringsgränssnittet är baserat på XML-meddelanden som används för att överföra information mellan RAIsoft-LTC och de övriga applikationerna. Genom att använda meddelandebaserad kommunikation så kan man göra förfrågningar på flera skalor samtidigt genom ett enda anrop till gränssnittet. XML är en mycket använd standard för meddelandeformat inom hälsovården (se kapitel 3). Således är det många systemleverantörer inom branschen som har, åtminstone i viss mån, använt XML och därför lämpar det sig som meddelandeformat för detta gränssnitt. XML är visserligen en omfattande teknologi och för ett enkelt gränssnitt som detta skulle man klara sig med ett mindre, skräddarsytt meddelandeformat. Man skulle dock i så fall gå mer mot en

[5] Med skalor menas de värden som beräknas av RAIsoft-LTC och som beskriver patientens tillstånd. Dessa skalor är en del av den kliniska patientinformation som är specifik för RAIsoft-LTC.

skräddarsydd lösning samt förlora den utvidgningsbarhet som XML erbjuder, detta strider mot de återanvändningskrav som ställs i kravspecifikationen (se kapitel 4).

Som mellanprogramsteknologi för gränssnittet används Web Services. Web Services är en teknologi genom vilken man kan erbjuda ett integreringsgränssnitt, eller en tjänst, som övriga applikationer kan komma åt genom en webbserver [Whatis 2005a]. Eftersom Web Services bygger på webbteknologi så är de tjänster som erbjuds åtkomliga i de omgivningar där webbteknologi är i användning. Detta innebär att Web Services inte är beroende av vilken plattform de olika applikationerna kör på, så länge som plattformen stöder webbteknologi. [Newcomer 2002] I denna försökstillämpning används Apache som webbserver, Web Service-anropen skickas vidare från webbservern till adapterskiktet i integreringsgränssnittet. Försökstillämpningens uppläggning är illustrerad i bild 9.

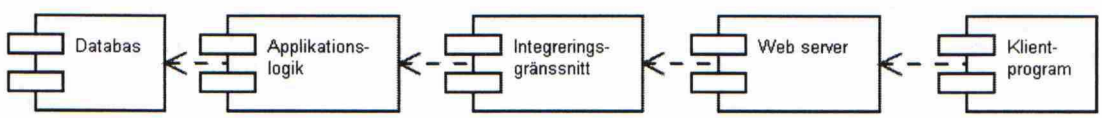

Bild 9 Komponentdiagram över försökstillämpningen.

6.2 *Integreringsgränssnittet*

Integreringsgränssnittet är uppbyggt enligt modellen som beskrivits i kapitel 5, det har alltså indelats i två skikt: API-skiktet och adapterskiktet. Utöver dessa två skikt så använder sig gränssnittet av applikationslogik som RAIsoft-LTC erbjuder. Denna applikationslogik kan också ses som ett separat skikt, funktionalitetsskiktet. Dessa tre nämnda skikt beskrivs närmare nedan.

Funktionalitetsskiktet

I funktionalitetsskiktet finns de funktionaliteter som är gemensamma för klientprogramvaran i RAIsoft-LTC samt detta gränssnitt. Detta skikt finns implementerat i klassen DataModule (se klassdiagrammet nedan). För att kunna erbjuda ett gränssnitt med patienters skalor behövs det två olika funktionaliteter: autentisering av användaren och hämtning av patienternas skalor. Dessa

funktionaliteter har kopierats från RAIsoft-LTCs programkod och bearbetats till ett programbibliotek som erbjuder två funktioner:

authenticateUser(userName, password, SSN): authenthicationResult
Denna funktion motsvarar funktionen authenticateUser som beskrivs i autentiseringsmodulen, kapitel 5. Här ges alltså användarnamn, användarens lösenord samt patientens socialskyddssignum som parametrar. Som returvärde får man information om huruvida användaren har tillgång till patientens uppgifter.

getScales(SSN, scales[]): scaleValue[]
Denna funktion motsvarar funktionen getRaisoftPI som beskrivs i modulen för patientinformation, kapitel 5. Som parametrar ges patientens socialskyddssignum samt en lista på de skalor vars värde man vill få som svar. Som returvärde ges en lista med värdena på de skalor som givits som parameter.

Detta skikt är alltså implementerat separat i ett programbibliotek. Enligt modellen i kapitel 5 ska även klienten i RAIsoft-LTC använda sig av detta programbibliotek för dessa funktionaliteter. Detta har dock inte gjorts inom ramen för denna försökstillämpning, det skulle dock vara nästa steg att göra när man följer modellen i kapitel 5.

API-skiktet

I detta skikt finns, enligt modellen i kapitel 5, de funktioner som erbjuds åt andra applikationer. Detta gränssnitt är implementerat i klassen MessageHandler. I försökstillämpningens gränssnitt finns det endast en funktion som erbjuds åt andra applikationer:

processScalesRequest(XMLRequest): string
Denna funktion tar ett meddelande som parameter och genererar ett annat meddelande som svar. Båda meddelandena är XML-dokument. Här läser alltså funktionen in meddelandet som innehåller en förfrågan och med denna information kallar den på de funktioner som funktionalitetsskiktet erbjuder. När patientinformationen så har hämtats, genereras ett svarsmeddelande med denna information.

Adapterskiktet

API-skiktet erbjuder alltså den funktion som övriga applikationer kan kalla på. Funktionen finns dock i det skiktet ännu på en nivå som ett program endast internt kan kalla på. Adapterskiktet implementerar det gränssnitt som övriga applikationer får tillgång, i detta fall en Web Service, och ger på så vis kopplingen från övriga applikationer till integreringsskiktet. Detta skikt implementeras här av två klasser: WebModule och Scales. Dessutom finns det en gränssnittsdefinition: IScales, som beskriver det gränssnitt som erbjuds.

Klassen WebModule implementerar den funktionalitet som behövs för att kunna utföra Web Service tjänster. IScales definierar det gränssnitt som erbjuds i Web Service-gränssnittet. Detta gränssnitt består av endast en funktion:

getScales (XMLRequest): string

Klassen Scales implementerar gränssnittet IScales och innehåller alltså samma funktion som finns i IScales definition. Denna funktion skickar vidare sin parameter till API-klassens processScalesRequest-funktion och returnerar det returvärde som API-klassens funktion returnerar.

Klassdiagrammet i bild 10 beskriver uppläggningen av integreringsgränssnittet.

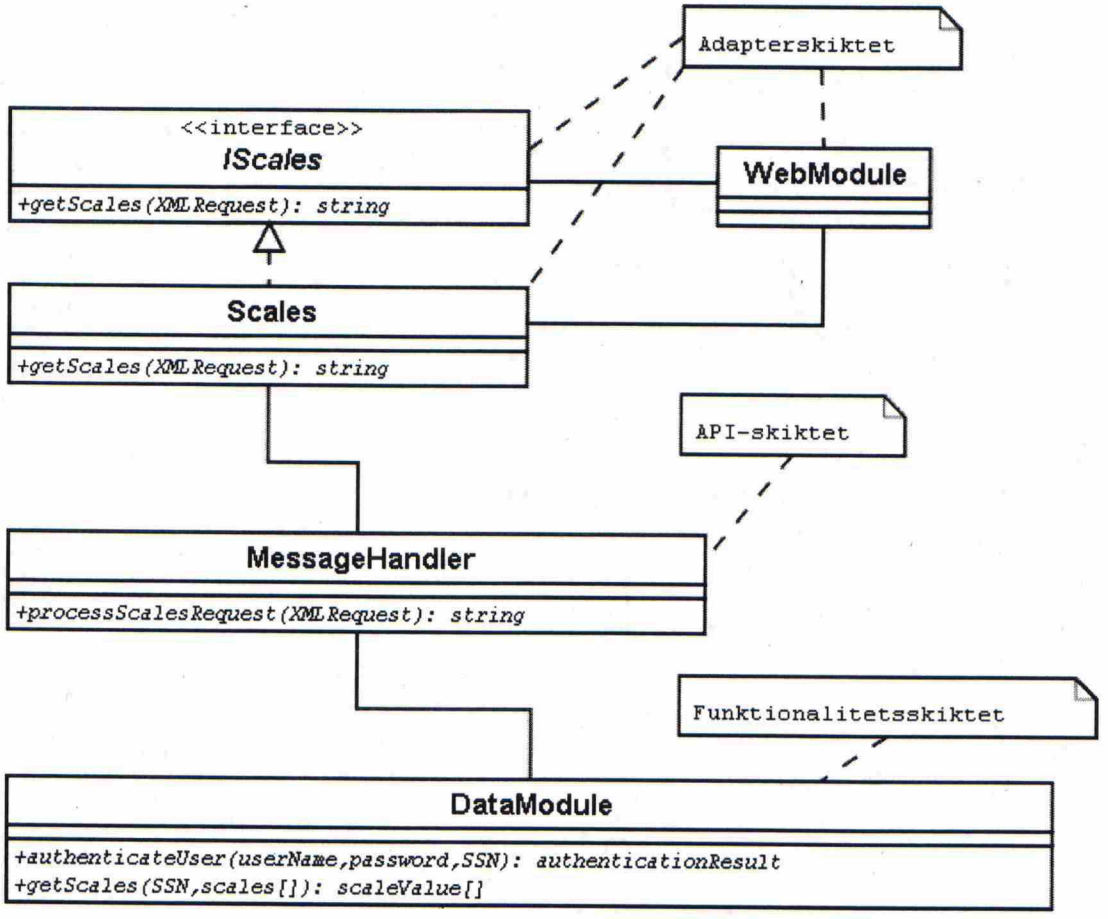

Bild 10 Klassdiagram över integreringsgränssnittet i försökstillämpningen.

6.3 Klientprogrammet

Klientprogrammet är ett litet program vars syfte endast är att testa det ovan beskrivna integreringsgränssnittet. I detta program kan användaren ge användarnamn, lösenord, patientens socialskyddssignum samt välja vilka skalor denne vill begära från integreringsgränssnittet. I användargränssnittet kan man även ange vilken URL integreringsgränssnittets Web Service finns på. När användaren har fyllt i denna information så skapar programmet ett XML-meddelande med denna information och skickar detta meddelande till Web Service tjänsten som en förfrågan. Denna förfrågan sker genom tjänstens *getScales*-funktion som finns definierad i *IScales* (se ovan). Som svar på denna förfrågan fås värdena på de skalor som begärts. Bilden nedan visar användargränssnittet för klientprogrammet.

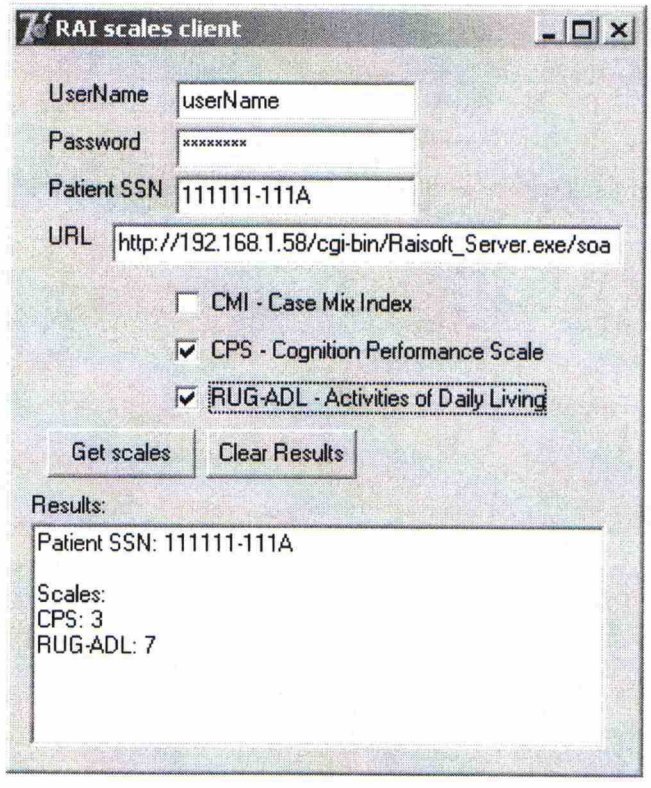

Bild 11 Försökstillämpningens klientprogram.

6.4 Utvärdering av försöksapplikationen

6.4.1 Testning av tillämpningen

För att få en överblick om hur bra integreringsgränssnittet fungerar så gjordes tidtagningar på förfrågningar som gjordes av klientprogrammet mot integreringsgränssnittet. Detta skedde så att klientprogrammet loggade tiden det tog för att utföra en förfrågan. På samma sätt loggade integreringsgränssnittet både tiden det tog från att man mottog en förfrågan till att man skickade svaret på denna förfrågan, och den tid funktionalitetsskiktet, dvs. applikationslogiken, använde under en förfrågan. Omgivningen för detta test såg ut som följande: databasen, integreringsgränssnittet och webbservern fanns på en dator medan klientprogrammet kördes på en annan dator som var kopplad ihop med serverdatorn genom ett Ethernet nätverk. Klientprogrammet automatiserades så att det gjorde 1000 olika förfrågningar på slumpmässigt valda patienter. I testet användes en databas med ca 10000 patienter. Resultaten från testet ser ut som följande:

- Medelvärdet för tiden som går åt till en förfrågning, från det att klientprogrammet skickar iväg förfrågningsmeddelandet tills det har mottagit svarsmeddelandet: *0,90 sekunder*
- Medelvärdet för tiden det tar för API-skiktet att behandla en förfrågan, inklusive applikationslogiken: *0,78 sekunder*
- Medelvärdet för tiden det tar funktionalitetsskiktet, dvs. applikationslogiken, att sköta autentisering och hämta data från databasen: *0,62 sekunder*

Från dessa resultat kan man dra den slutsatsen att det går i medeltal åt *0,90 – 0,78 = 0,12 sekunder* till att skicka meddelandena mellan klientprogrammet, webbservern och integreringsgränssnittet. Man kan även konstatera att integreringsgränssnittets API- och adapterskikt använder i medeltal *0,78 – 0,62 = 0,16 sekunder* till att hantera meddelanden, dvs. att ta emot, avkoda, generera samt sända de meddelanden som hör till en förfrågan. Se nedanstående tabell för en sammanfattning av testet.

	Tid (sekunder)	Procent av hela förfrågan
Överföring	0,12	13 %
Meddelandehantering	0,16	18 %
Applikationslogik	0,62	69 %
Totalt	0,90	100 %

Tabell 1 Resultat vid testning av försökstillämpningen.

I den ovanstående tabellen kan man se att den största delen av tiden för en förfrågan (69 %) går åt till applikationslogiken. Det att dataöverföring och meddelandehanteringen tar upp en relativt liten del av hela förfrågan (totalt 0,28 sekunder eller 31 %) ger en antydan om att dessa funktioner inte tar upp alltför mycket resurser. Ifall dataöverföringen och meddelandehanteringen upptog större delen av tiden för en förfrågan, jämfört med applikationslogiken, så skulle detta orsaka en flaskhals som skulle göra systemet långsamt, speciellt vid integrering med större omfattning.

Dessa testresultat beror dock mycket på i hurudan omgivning integreringsgränssnittet är i bruk. Databasens egenskaper, webbserverns belastning, serverdatorns och nätverkets egenskaper samt belastning har inverkan på hur länge det tar att utföra en

förfrågan. Därför är dessa testresultat endast riktgivande och kan inte tolkas som absoluta värden för ett dylikt system i produktionsomgivning.

6.4.2 Fördelar och nackdelar med modellen

Genom att implementera modellen i denna försökstillämpning har det kommit fram hur modellen fungerar i praktiken. Nedan är en lista på både för- och nackdelar:

Fördelar:

- Denna modell är delad upp i flera olika skikt, vilket underlättar överblicken av programmets uppläggning. Detta underlättar implementering av modellen samt vidareutveckling av programkoden.
- I och med att mellanprogramsteknologin är implementerad i ett separat skikt, adapterskiktet, så är det enkelt att lägga till stöd för en annan mellanprogramsteknologi som, t.ex. CORBA eller COM. Detta är alltså möjligt utan att behöva röra programkoden i de övriga skikten.
- Eftersom meddelandehanteringen även finns i ett separat skikt, API-skiktet, går det även enkelt att lägga till stöd för ett annat meddelandeformat utan att behöva göra några stora ändringar i adapterskiktet.

Nackdelar:

- Funktioner i adapterskiktet bidrar egentligen inte med någon funktionalitet, utan kallar endast på funktioner ur API-skiktet. Funktionen *getScales* i *Scales*-klassen i adapterskiktet gör egentligen ingenting annat än kallar på *processScalesRequest*-funktionen i klassen *MessageHandler* i integreringsskiktet. Detta kan bidra till att programkoden blir ineffektiv. Dessa två olika funktioner är dock nödvändiga för att kunna särskilja adapter- och API-skiktet.
- Klienten i RAIsoft-LTC och integreringsgränssnittet som båda ska använda sig av samma programbibliotek har olika behov. Det innebär att det krävs mycket resurser i planering för att programbiblioteket ska vara så generellt att det lämpar sig både för klienten som användaren brukar på sin arbetsstation och för integreringsgränssnittet.
- I denna försökstillämpning är det enkelt att avlyssna datatrafiken som går över nätverket och på så vis få tag på patientuppgifter samt användares lösenord. Detta kräver alltså att organisationen som använder denna lösning

måste se till att nätverksförbindelserna är säkra, genom att t.ex. använda krypterad nättrafik. I detta fall med Web Services kunde Transport Layer Security (TLS) [Whatis 2005b] vara ett alternativ för att göra nätförbindelsen säker.

En styrka i denna modell är en enkel uppdelning i två skikt. Det gör implementering av modellen lätt, samtidigt som det är enkelt att utvidga lösningen med stöd för en ny mellanprogramsteknologi eller meddelandestandard. Eftersom funktioner i adapterskiktet, som det redan nämndes ovan, inte direkt bidrar med någon funktionalitet, så föreligger det dock en risk att man när man vidareutvecklar integreringsgränssnittet slår ihop funktioner från dessa två skikt. Detta leder dock till att man förlorar gränsen mellan adapter- och integreringsskiktet och på så vis försämrar möjligheterna att återanvända programkoden. Därför är det viktigt att hålla fast vid skiktindelningen fastän funktionerna i adapterskiktet vid första anblick inte verkar bidra med någon annan funktionalitet än att de kallar på motsvarande funktioner i API-skiktet.

7 Slutsatser

7.1 Tillvägagångssätt vid integrering

Vid första anblick är det lätt hänt att man ser integrering av informationssystem som ett rent tekniskt problem. Dvs. att man vill utarbeta en teknisk lösning som gör det möjligt för två eller flera applikationer att utbyta information. Men när man ser närmare på den teknologi som finns tillgänglig för systemintegrering så kan man konstatera att det redan finns en hel del färdig teknologi att fås. Denna teknologi kan användas som sådan eller vidareutvecklas för att stöda den tekniska lösning man tar i bruk. Även om det tar tid och pengar att utveckla den tekniska lösningen så finns det alltså färdig teknologi som man kan använda sig av och man behöver inte bygga upp hela den tekniska lösningen från noll. I det stora hela är alltså den tekniska delen av integreringen ingen oöverkomligt svår uppgift.

Kärnan i den problemställning som uppkommer vid systemintegrering ligger i planering och förhandlingar mellan de olika parterna som deltar i integreringen. På detta plan handlar integrering om att komma överens om hur man vill gå tillväga för att få integreringen i användning. Här har både systemleverantörerna och kunden olika intressen som till en del kan strida mot varandra. Det gäller att komma överens om på vilken nivå man vill ta integreringen i bruk, på datanivån, applikationsnivån, affärsprocessnivån eller en kombination av dessa. Det gäller också att komma överens om hurudan teknologi och vilka standarder man vill använda för den aktuella integreringslösningen. När man tar dessa beslut är det viktigt att även ta framtida utsikter för informationssystemen i beaktande. Man måste avgöra om man vill satsa resurser på en integreringslösning, som även stöder framtida behov, eller om man går in för en mindre kostsam lösning som är anpassad enligt de behov som man har just nu. De integreringslösningar som kundens egna samarbetspartners använder kan också ha en stor roll, speciellt om integreringslösningen går utanför kundens egen organisation.

När man väl kommit så här långt i planeringen är det en relativt lätt uppgift att implementera den lösning som planerats. Tyngdpunkten vid den problemställning som systemintegrering utgör ligger alltså vid hurudan policy man går in för när man

bygger upp en integreringslösning. De rent tekniska problemen spelar givetvis en roll, men dessa är lättare att lösa när man väl kommit överens om att gå in för en viss policy.

7.2 Forskningsresultat

Den kartläggning av integrering som gjordes i den första delen av detta diplomarbete har gett en översikt över hur man går tillväga vid integrering inom organisationer. Kartläggningen gav även en översikt över vilka teknologier som används inom integrering av hälsovårdsrelaterade informationssystem. Denna översikt var en viktig förutsättning för den andra delen av diplomarbetet. Utan denna kartläggning skulle det vara svårt att veta vilka faktorer som skall tas i beaktande vid planering av integreringsmodellen för RAIsoft-LTC. Kartläggningen kommer även att ha betydelse när modellen tillämpas i praktiken, den information som kartläggningen bidrar med gör det lättare för Raisoft som systemleverantör att se vilka faktorer som ligger i Raisofts intresse vid integrering.

Den modell för integrering av RAIsoft-LTC som presenteras i den andra delen av detta diplomarbete är gjort med återanvändning som utgångspunkt. Eftersom modellen inte var gjord för en enskild lösning så är den inte heller begränsad till någon viss teknologi eller standard. Avsikten med detta var att kunna återanvända modellen i olika integreringslösningar och att kunna anpassa modellen efter den policy som de olika integreringslösningarna följer. Försökstillämpningen som gjordes för att testa modellen verifierar att modellen är lätt att tillämpa med olika teknologier.

7.3 Förslag till vidareutveckling av integreringsmodellen

I integreringsmodellen som utarbetats för RAIsoft-LTC föreslås att applikationens klientprogram skall använda sig av ett programbibliotek som är gemensamt med integreringsgränssnittet. Hur programbiblioteket är uppbyggt tas inte upp här. Denna uppgift är dock nödvändig för att återanvända programkod enligt modellen, således kunde ett sådant programbibliotek planeras som fortsättning på detta diplomarbete.

Integreringsmodellen för RAIsoft-LTC kunde också vidareutvecklas för att användas inom Raisofts övriga programvaror. RAIsoft-LTC, som används inom

långtidsvården, hör till samma produktfamilj som t.ex. RAIsoft-HC som används inom hemvården. Användningen av informationssystem inom hemvården skiljer sig till en del från användningen av informationssystem inom långtidsvården. Detta leder till att det finns faktorer som är speciella för hemvården och som borde tas i beaktande för att integreringsmodellen ska kunna användas för RAIsoft-HC.

8 Källförteckning

[Culpepper 2000] Culpepper T. (2000). National Committee on Vital and Health Statistics: Subcommittee on Standards and Security. [Refererad 27.11.2004]. http://www.ncvhs.hhs.gov/000713s01.pdf

[Farrell 2002] Farrell M., Lublinsky B. (2002). Top 10 Reasons Why EAI Fails. EAI Journal, december 2002. [Refererad 8.11.2003]. http://www.bijonline.com/PDF/LublinskyEAIFails.pdf

[Finlex 2004] Statens författningsdata - FINLEX. Lag om patientens ställning och rättigheter 17.8.1992/785. [Refererad 16.9.2004]. http://www.finlex.fi/sv/laki/alkup/1992/19920785

[Gormly 2001] Gormly L. (2001). EAI Overview. [Refererad 8.11.2003]. http://eai.ittoolbox.com/documents/document.asp?i=1246

[Grudén 2003] Grudén A., Strannegard P. (2003). Business Process Integration: The Next Wave. EAI Journal, januari 2003. [Refererad 13.11.2003]. http://www.bijonline.com/PDF/Jan03Grueden.pdf

[Hautsalo 2002] Hautsalo A., Häyrinen K., Korhonen M. (2002). Terveydenhuollon tietojärjestelmien yhteensopivuus – kaukainen tavoite vai pian todellisuutta. Kuntapuntari 3/2002. [Refererad 10.9.2004]. http://www.plugit.fi/julkaisut/docs/Hautsalo-Hayrinen-Korhonen-2002.pdf

[HL7 2004a] About Health Level Seven. [Refererad 8.10.2004]. http://www.hl7.org/about/

[HL7 2004b] HL7 Finland ry. (2004). Open CDA määrittelydokumentti. [Refererad 12.10.2004]. http://www.hl7.fi/opencda.zip

[HL7 2004c] HL7 Worldwide. [Refererad 8.10.2004]. http://www.hl7.org.uk/marketing/hl7worldwide.asp

[HL7 2004d] HL7 Reference Information Model. [Refererad 22.11.2004]. http://www.hl7.org/library/data-model/RIM/C30204/rim.htm

[Korpela 2004] Korpela M. (2004). PlugIT:n tulokset pähkinänkuoressa. [Refererad 28.11.2004]. http://www.plugit.fi/docs/esitys/plugit-yhteenveto-021104.ppt

[Kärkkäinen 2001] Kärkkäinen S., Maunuksela-Malinen P., Saloranta A. (2001). Yritysten välinen sähköinen liiketoiminta - EDI/OVT:n käyttö Suomessa. Helsingfors. TIEKE Tietotekniikan kehittämiskeskus ry.

[Köyste 2002] Köyste A. (2002). Kotimaiset hyväksytyt EDIFACT-sanomat. [Refererad 7.12.2004]. http://www.tieke.fi/sahkoinen.nsf/

[Linthicum 2001] Linthicum D. (2001). B2B Application Integration: e-Business-Enable Your Enterprise. Addison-Wesley.

[Linthicum 2004] Linthicum D. (2004). Next Generation Application Integration: From Simple Information to Web Services. Addison-Wesley.

[Mann 1999] Mann J. (1999). Workflow and EAI. EAI Journal, september/oktober 1999. [Refererad 8.11.2003]. http://www.bijonline.com/PDF/mann_1.pdf

[Newcomer 2002] Newcomer E. (2002). Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley.

[OMG 1998a] Object Management Group (1998). Lexicon Query Service RFP Response. [Refererad 24.11.2004]. http://www.omg.org/docs/corbamed/98-01-09.pdf

[OMG 1998b] Object Management Group (1998). Person Identification Service. [Refererad 24.11.2004]. http://www.omg.org/docs/corbamed/98-01-09.pdf

[OMG 1999] Object Management Group (1999). Resource Access Decision. [Refererad 24.11.2004]. http://www.omg.org/docs/corbamed/99-04-04.pdf

[OMG 2000a] Object Management Group (2000). Clinical Image Access Service. [Refererad 24.11.2004]. http://www.omg.org/docs/corbamed/00-02-01.pdf

[OMG 2000b] Object Management Group (2000). Clinical Observations Access Service Specification. [Refererad 24.11.2004]. http://www.omg.org/docs/dtc/00-01-01.pdf

[OMG 2004] Object Management Group Healthcare Domain Task Force - Executive summary. [Refererad 24.11.2004]. http://healthcare.omg.org/Healthcare_files/Executive_Summary.html

[PlugIT 2004] PlugIT Rajapinnat. [Refererad 28.11.2004]. http://www.plugit.fi/rajapinnat/

[RAI 2005] RAI presentation. [Refererad 12.1.2005]. http://www.finrai.org/

[Reese 2002] Reese A. (2002). Enterprise Application Integration 101. [Refererad 8.11.2003]. http://www.isourceonline.com/article.asp?article_id=2922

[Sadoski 2004] Sadoski D. (2004). Two Tier Software Architectures. [Refererad 14.10.2004]. http://www.sei.cmu.edu/str/descriptions/twotier.html

[Savolainen 2004] Savolainen K. (2004). EDI ja standardointi. [Refererad 7.12.2004]. http://www.tieke.fi/sahkoinen.nsf/

[Schmidt 2002] Schmidt J. (2002). The EAI Laws. EAI Journal, juli 2002. [Refererad 8.11.2003]. http://www.bijonline.com/PDF/EAILaws.pdf

[Seliger 2001] Seliger R. (2001). Overview of HL7's CCOW Standard. [Refererad 24.11.2004]. http://www.hl7.org/library/committees/sigvi/ccow_overview_2001.doc

[Stultz 2001] Stultz R. (2001). Demystifying EDI. Wordware Publishing Inc.

[Tervo-Pellikka 1996] Tervo-Pellikka R. (1996). Terveydenhuollon tietojärjestelmät: suomalaisten standardien kehittäminen. Helsingfors. Oy Edita Ab.

[Tuomainen 2004] Tuomainen M. (2004). Kontekstinhallinnan määrittely versio 2. [Refererad 28.11.2004]. http://www.plugit.fi/docs/proj/Kontekstinhallinta-v2.pdf

[Whatis 2004a] Whatis.com sökmotor, sökord: DTD. [Refererad 10.11.2004]. http://whatis.techtarget.com/

[Whatis 2004b] Whatis.com sökmotor, sökord: XML. [Refererad 10.11.2004]. http://whatis.techtarget.com/

[Whatis 2005a] Whatis.com sökmotor, sökord: web service. [Refererad 2.1.2005]. http://whatis.techtarget.com/

[Whatis 2005b] Whatis.com sökmotor, sökord: tls. [Refererad 10.1.2005]. http://whatis.techtarget.com/